DISSERTATION

On

COLLABORATIVE FILTERING RECOMMENDATION ALGORITHMS TO ALLEVIATE SPARSITY AND SCALABILITY PROBLEMS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE

of

MASTER OF ENGINEERING

in

COMPUTER TECHNOLOGY & APPLICATIONS

By

Rakhi

(University Roll no. - 8552)

Under the Guidance of

Mrs. Akshi kumar

Assistant Professor

DEPARTMENT OF COMPUTER ENGINEERING

DELHI COLLEGE OF ENGINEERING

DELHI UNIVERSITY

2011

Certificate

This is to certify that the major project entitled **"Collaborative Filtering Recommendation Algorithms to alleviate Sparsity and Scalability problems"** is the work of Rakhi **(** Univ. Roll No. 8552) , a student of Delhi College of Engineering. This work was completed under my direct supervision and guidance and forms a part of Master of Engineering (Computer Technology & Applications) course and curriculum. She has completed her work with utmost sincerity and diligence.

(Mrs. Akshi kumar)
Project Guide
Department of Computer Engineering
Delhi Technological University

Acknowledgement

No volume of words is enough to express my gratitude towards my guide **Mrs. Akshi Kumar**, Assistant Professor, Computer Science and Engineering, Delhi Technological University, Delhi, who has been very concerned and has aided for all the materials essential for the preparation of this thesis report. She has helped me to explore this vast topic in an organized manner and provided me with all the ideas on how to work towards a research-oriented venture.

I am also thankful to **Dr. (Mrs) Daya Gupta**, Head of the Department, Computer Science and Engineering, Delhi Technological University, Delhi for the motivation and inspiration that triggered me for the thesis work.

I would also like to thank the staff members and my colleagues who were always there at the need of the hour and provided with all the help and facilities, which I required for the completion of my thesis work.

Most importantly, I would like to thank my parents and the almighty for showing me the right direction out of the blue, to help me stay calm in the oddest of the times and keep moving even at times when there was no hope

Rakhi

(Univ. Roll No. – 8552)

ABSTRACT

The Internet and World Wide Web have given us a world of endless possibilities-like items to consume, movies to watch, music to listen, conversations to participate in etc. Amidst all this range of endless options, a consumer faces the task of what to choose which might interest him. Recommender system comes to the rescue for such a consumer. These systems aim to mediate, support, or automate the everyday process of sharing recommendations.

Recommender systems are making their presence felt in a number of domains, be it for ecommerce or education, social networking etc. With huge growth in number of consumers and items in recent years ,recommender systems faces some key challenges. These are: producing high quality recommendations and performing many recommendations per second for millions of consumers and items. New recommender system technologies are needed to reduce sparseness in order to get high quality recommendations, even for very large-scale problems.

In this thesis, we focus on collaborative approach based recommender systems to solve the issues of sparsity and scalability. We have compared two collaborative filtering algorithms which solves the above mentioned issues in one go. The first algorithm uses weighted slope one method to reduce sparseness and the other one uses item classification technique. Then item clustering is used to alleviate the scalability problem. The item classification technique is better among the two as determined using the case study from movielens data set obtained from Grouplens website.

TABLE OF CONTENTS

LIST OF FIGURES

TABLE OF TABLES

CHAPTER 1

INTRODUCTION

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

The term "collaborative filtering" was introduced in the context of the first commercial recommender system, called Tapestry [1], which was designed to recommend documents drawn from newsgroups to a collection of users. It was created to leverage social collaboration in order to prevent users from getting inundated by a large volume of streaming documents. Collaborative filtering, which analyzes usage data across users to find well matched user-item pairs, has since been juxtaposed against the older methodology of content filtering which had its original roots in information retrieval. In content filtering, recommendations are not "collaborative" in the sense that suggestions made to a user do not explicitly utilize information across the entire user-base. Some early successes of collaborative filtering on related domains included the GroupLens[2] system. Initial formulations for recommender systems were based on straightforward correlation statistics and predictive modeling, not engaging the wider range of practices in statistics and machine learning literature. Several efforts have been attempted to combine content-based methods with collaborative filtering, and to incorporate additional domain knowledge in the architecture of recommender systems.

Further research was spurred by the public availability of datasets on the web, and the interest generated due to direct relevance to e-commerce. Netflix, an online streaming video and DVD rental service, released a large-scale dataset containing 100 million ratings given by about half-a-million users to thousands of movie titles, and announced an open competition for the best collaborative filtering algorithm in this domain.

More formally, the recommendation problem can be formulated as follows: Let C be the set of all users and let S be the set of all possible items that can be recommended, such as books, movies, or restaurants. The space S of possible items can be very large, ranging in hundreds of thousands or even millions of items in some applications, such as recommending books or CDs. Similarly, the user space can also be very large—millions in some cases. Let u be a utility function that measures the usefulness of items to user c, i.e., $u : C \times S \longrightarrow R$, where R is a totally ordered set (e.g., nonnegative integers or real numbers within a certain range). Then, for

each user c □ C, we want to choose such item s' □ S that maximizes the user's utility. In recommender systems, the utility of an item is usually represented by a rating, which indicates how a particular user liked a particular item.

The central problem of recommender systems lies in that utility u is usually not defined on the whole C × S space, but only on some subset of it. This means u needs to be extrapolated to the whole space C × S. In recommender systems, utility is typically represented by ratings and is initially defined only on the items previously rated by the users. For example, in a movie recommendation application (such as the one at MovieLens.org), users initially rate some subset of movies that they have already seen. An example of a user-item rating matrix for a movie recommendation application is presented in Table 1, where ratings are specified on the scale of 1 to 5. The "Ø" symbol for some of the ratings in Table 1 means that the users have not rated the corresponding movies. Therefore, the recommendation engine should be able to estimate (predict) the ratings of the nonrated movie/user combinations and issue appropriate recommendations based on these predictions.

	K-PAX	Life of Brian	Memento	Notorious
Alice	4	3	2	4
Bob	Ø	4	5	5
Cindy	2	2	4	Ø
David	3	Ø	5	2

TABLE 1 A Fragment of a Rating Matrix for a Movie Recommender System

Extrapolations from known to unknown ratings are usually done by

1) specifying heuristics that define the utility function and empirically validating its performance.

2) estimating the utility function that optimizes certain performance criterion, such as the mean square error.

Once the unknown ratings are estimated, actual recommendations of an item to a user are made by selecting the highest rating among all the estimated ratings for that user, according to (1). Alternatively, we can recommend the N best items to a user or a set of users to an item [3].

In this age of information overload, people use a variety of strategies to make choices about what to buy, how to spend their leisure time, and even whom to date. Recommender systems automate some of these strategies with the goal of providing affordable, personal, and high-quality recommendations. A good recommendation engine is worth a lot of money. According to a report by industry analyst Forrester, one-third of customers who notice recommendations on an e-commerce site wind up buying something based on them. The trouble with recommendation engines is that they're really hard to build. Though they look simple but they're actually doing something fiendishly complex. They're processing astounding quantities of data and doing so with seriously high-level math. That's because they're attempting to second-guess a mysterious, perverse and profoundly human form of behavior: the personal response to a work of art. They're trying to reverse-engineer the soul. They're also changing the way our culture works. We used to learn about new works of art from friends and critics and video-store clerks — from people, in other words. Now we learn about them from software. There's a new class of tastemakers, and they're not human.

Currently, Recommender Systems remain an active area of research. Applications have been pursued in diverse domains ranging from recommending web pages to music, books, movies and other consumer products.

While studying recommender systems, the author learnt that a good recommendation system needs to address many challenges. Major among them are: Data sparsity, Scalability, Shilling Attacks, Synonymy etc. Dealing with data sparsity issue and need to improve scalability is the major concern for any recommendation system providing accurate recommendations i.e. predictions which closely matches with user interest. So the author has focused mainly to resolve sparsity and scalability issues.

1.2 Research Objective

Data sparsity means that in a user – item rating database, few ratings are available as compared to the number of items available for the users. So in such a sparse dataset, accuracy is not high if predictions are done using any of the existing item based or user based collaborative filtering methods. Scalability is another major issue with the size of the data set being enormously huge. It becomes difficult to search the entire rating database and there is poor scalability when more and more users and items are added into the database. Also scalability cannot be improved much in memory based and item based collaborative filtering methods. Keeping this in mind, we have analyzed two collaborative filtering algorithms, which deals with the above two mentioned issues simultaneously. The problem statement is:

"To find out which collaborative filtering algorithm is better suited to deal with sparsity and scalability issues – a collaborative filtering algorithm based on weighted slope one scheme and item clustering or a collaborative filtering algorithm based on item classification and item clustering. We are determining which is better in terms of simplicity and accuracy among the two methods. "

1.3 Proposed Work

In this thesis, we have compared two collaborative filtering algorithms tackling the issues of sparsity and scalability. The first algorithm uses weighted slope one scheme technique to reduce the sparsity of the data set and the other algorithm uses the item classification technique for the same. The weighted slope one scheme is based on popularity differential of items i.e. in a pair wise fashion it is determined which item is liked better than other. This information is further used to predict rating for a user given their rating for other items. This is the idea behind slope one scheme. But since this scheme does not consider number of ratings observed so weighted slope one scheme is used which consider this information too. The item classification technique classifies items (here movies) into groups based on some attribute. The attribute chosen here is genre. The various genres that movies could belong to are Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western. So in all we have divided the collection of all movies in 18 groups. Further then in each of the group user based collaborative filtering is used

to predict ratings. Since a movie could belong to various genres in that case mean of values predicted from each group is used to get final rating for that movie. Then item clustering (K-means) have been performed on the dense data set obtained from each of the techniques. Using the item clustering items are clustered into groups.48 clusters are created. Clustering results in similar items grouped together. The item for which rating needs to be predicted is matched with the centroid of each of the clusters. Then neighbors in the nearest matching centroids is considered as neighbors for the item in question. Once neighbors are identified weighted average weighted by similarity is used to determine rating. The dense data set in each of the above cases is divided into base file and test file. 5% of the users are considered as the test users. Finally the results have been compared for both the algorithms using Mean Absolute Error as the evaluation metric. For calculating MAE, predictions were withheld for 5% of the users and then predicted rating difference with the withheld rating is used to determine MAE. It turns out that item classification technique scores over weighted slope one scheme in terms of accuracy of predictions. Also it is easier to implement and computation time is lesser for creating the dense data set since predictions are done within groups.

1.4 Organization of Remainder Of Thesis

In the above chapter, we had discussed the motivation, research objective and finally the proposed work. Chapter 2 provides the literature review of various types of recommender systems and their evaluation metrics. Chapter 3 has the complete details about the techniques employed to compare two algorithms and how the techniques are combined to resolve sparsity and scalability challenges. Chapter 4 shows the experimental setup and results of the algorithms compared, followed by limitations and finally chapter 5 consists of the contributions and possible future work or directions in this area and it finally ends with the references and appendices.

1.5 Summary

In this chapter we have thrown some insight into the field of recommender system followed by the motivation for the author to do this work. Finally author had described the objectives of the

thesis and how he accomplishes these objectives in his research using certain sparsity reducing techniques and item clustering algorithm.

CHAPTER 2

LITERATURE SURVEY

CHAPTER 2

LITERATURE SURVEY

2.1 Web 2.0

The term **Web 2.0** is associated with web applications that facilitate participatory information sharing, interoperability, user-centered design, and collaboration on the World Wide Web. A Web 2.0 site allows users to interact and collaborate with each other in a social media dialogue as creators (prosumers) of user-generated content in a virtual community, in contrast to websites where users (consumers) are limited to the passive viewing of content that was created for them. Examples of Web 2.0 include social networking sites, blogs, wikis, video sharing sites, web applications, and folksonomies.

The term is closely associated with Tim O'Reilly because of the O'Reilly Media Web 2.0 conference in late 2004. Web 2.0 websites allow users to do more than just retrieve information. By increasing what was already possible in "Web 1.0", they provide the user with more user-interface, software and storage facilities, all through their browser. This has been called "Network as platform" computing. Users can provide the data that is on a Web 2.0 site and exercise some control over that data. The characteristics of Web 2.0 are: rich user experience, user participation, dynamic content, metadata, web standards and scalability. Further characteristics, such as openness, freedom and collective intelligence by way of user participation, can also be viewed as essential attributes of Web 2.0.

The client-side/web browser technologies used in Web 2.0 development are Asynchronous JavaScript and XML (Ajax), Adobe Flash and the Adobe Flex framework, and JavaScript/Ajax frameworks such as Yahoo! UI Library, Dojo Toolkit, MooTools, and jQuery.

Web 2.0 can be described in 3 parts which are as follows:

- Rich Internet Application (RIA) - It defines the experience brought from desktop to browser whether it is from a graphical point of view or usability point of view. Some buzzwords related to RIA are Ajax and Flash.

- Service oriented architecture(SOA) - It is a key piece in Web 2.0 which defines how Web 2.0 applications expose its functionality so that other applications can leverage and integrate the functionality providing a set of much richer applications (Examples are: Feeds, RSS, Web Services, Mash-ups)
- Social Web — It defines how Web 2.0 tend to interact much more with the end user and making the end-user an integral part.

An important part of Web 2.0 is the social Web, which is a fundamental shift in the way people communicate. The social web consists of a number of online tools and platforms where people share their perspectives, opinions, thoughts and experiences. Web 2.0 applications tend to interact much more with the end user. As such, the end user is not only a user of the application but also a participant

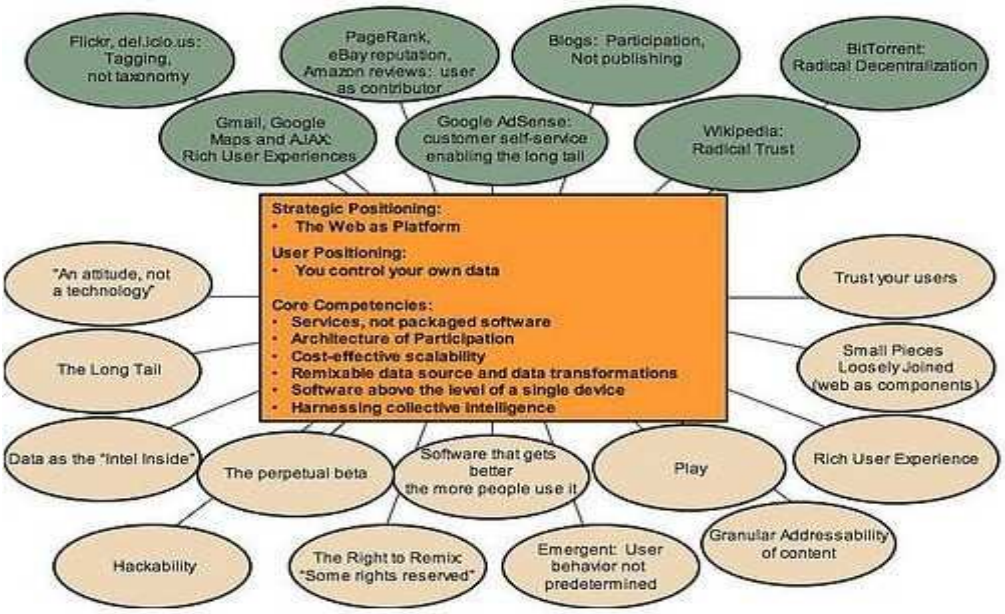

Figure 1: Web 2.0 Map

The Social Web provides huge opportunities for recommender technology and in turn recommender technologies can play a part in fuelling the success of the Social Web phenomenon. The social web has turned information consumers into active contributors creating massive amounts of information. Finding relevant and interesting content at the right time and in the right context can be one of the tasks of recommender systems.

Recommendation can be for anything like books, music, persons, news, web pages etc. One important example of use of recommendation engine is in the field of social shopping. Recommendation engines allow shoppers to provide advice to fellow shoppers. Traditional online product review companies such as Amazon have helped many consumers to date but currently emphasize obtaining and giving advice to strangers. Upcoming social shopping startups such as ShopSocially , Blippy , and Swipely now encourage conversations around purchases with a user's friends or acquaintances. Still there are many more unexplored fields which will surely benefit by making use of recommendation engines.

2.2 Recommender systems & Types

Recommender systems, **recommendation systems**, **recommendation engines**, **recommendation frameworks**, **recommendation platforms** work from a specific type of information filtering system technique that attempts to recommend information items (movies, TV program, video on demand, music, books, news, images, web pages, scientific literature such as research papers etc.) that are likely to be of interest to the user.

Typically, a recommender system compares a user profile to some reference characteristics, and seeks to predict the 'rating' that a user would give to an item they had not yet considered. These characteristics may be from the information item (the content-based approach) or the user's social environment (the collaborative filtering approach).When building the user's profile a distinction is made between explicit and implicit forms of data collection[4].

Examples of explicit data collection include the following:

- Asking a user to rate an item on a sliding scale.
- Asking a user to rank a collection of items from favorite to least favorite.
- Presenting two items to a user and asking him/her to choose the best one.
- Asking a user to create a list of items that he/she likes.

Examples of implicit data collection include the following:

- Observing the items that a user views in an online store.

- Analyzing item/user viewing times.
- Keeping a record of the items that a user purchases online.
- Obtaining a list of items that a user has listened to or watched on his/her computer.
- Analyzing the user's social network and discovering similar likes and dislikes [4].

The recommender system compares the collected data to similar and not similar data collected from others and calculates a list of recommended items for the user. Recommender systems are a useful alternative to search algorithms since they help users discover items they might not have found by themselves.

In its most common formulation, the recommendation problem is thus reduced to the problem of estimating ratings for the items that have not been seen by a user. Intuitively, this estimation is usually based on the ratings given by this user to other items and on some other information that will be formally described below. Once we can estimate ratings for the yet unrated items, we can recommend to the user the item(s) with the highest estimated rating(s).

2.2.1 Content Based Filtering / Recommender systems

The user will be recommended items similar to the ones the user preferred in the past. In content-based recommendation methods, the utility u(c, s) of item s for user c is estimated based on the utilities u(c, s_i) assigned by user c to items s_i □ S that are "similar" to item s. For example, in a movie recommendation application, in order to recommend movies to user c, the content-based recommender system tries to understand the commonalities among the movies user c has rated highly in the past (specific actors, directors, genres, subject matter, etc.). Then, only the movies that have a high degree of similarity to whatever the user's preferences are would be recommended [3].

The content-based approach to recommendation has its roots in information retrieval, and information filtering research. Because of the significant and early advancements made by the information retrieval and filtering communities and because of the importance of several text-based applications, many current content-based systems focus on recommending items containing textual information, such as documents, Web sites (URLs), and Usenet news messages. The improvement over the traditional information retrieval approaches comes from the use of user profiles that contain information about users' tastes, preferences, and needs. The

profiling information can be elicited from users explicitly, e.g., through questionnaires, or implicitly—learned from their transactional behavior over time.

Content-based filtering systems have the following advantages:

1. They don't require data on other users and are away from new user cold-start and sparsity problems.

2. These systems are capable of recommending items to users with unique tastes.

3. Can provide explanations of recommended items by explicitly listing content features or descriptions that caused an item to be recommended [5].

Content-based recommender systems have several limitations:

1. *Limited Content Analysis*

Content-based techniques are limited by the features that are explicitly associated with the objects that these systems recommend. Therefore, in order to have a sufficient set of features, the content must either be in a form that can be parsed automatically by a computer (e.g., text) or the features should be assigned to items manually. While information retrieval techniques work well in extracting features from text documents, some other domains like multimedia data have an inherent problem with automatic feature extraction. Moreover, it is often not practical to assign attributes by hand due to limitations of resources [6].

2. *Overspecialization*

When the system can only recommend items that score highly against a user's profile, the user is limited to being recommended items that are similar to those already rated. For example, a person with no experience with Greek cuisine would never receive a recommendation for even the greatest Greek restaurant in town.

3. *New User Problem*

The user has to rate a sufficient number of items before a content-based recommender system can really understand the user's preferences and present the user with reliable

recommendations. Therefore, a new user, having very few ratings, would not be able to get accurate recommendations.

2.2.2 Collaborative Filtering Systems

Collaborative Filtering is the process of filtering or evaluating items using the opinions of other people. Unlike content-based recommendation methods, collaborative recommender systems (or collaborative filtering systems) try to predict the utility of items for a particular user based on the items previously rated by other users. More formally, the utility u(c, s) of item s for user c is estimated based on the utilities u (c_j, s) assigned to item s by those users c_j □ C who are "similar" to user c. For example, in a movie recommendation application, in order to recommend movies to user c, the collaborative recommender system tries to find the "peers" of user c, i.e., other users that have similar tastes in movies (rate the same movies similarly). Then, only the movies that are most liked by the "peers" of user c would be recommended.

There have been many collaborative systems developed in the academia and the industry. The Tapestry system relied on each user to identify like-minded users manually. GroupLens[8], Video Recommender, and Ringo[6] were the first systems to use collaborative filtering algorithms to automate prediction. Other examples of collaborative recommender systems include the book recommendation system from Amazon.com, the PHOAKS system that helps people find relevant information on the WWW, and the Jester system that recommends jokes.

a. *Core Concepts*

Though there exists variety of CF systems, we introduce the topic through MovieLens. MovieLens is a collaborative filtering system for movies. A user of MovieLens rates movies using 1 to 5 stars, where 1 is "Awful" and 5 is "Must See". MovieLens then uses the ratings of the community to recommend other movies that user might be interested in (Figure 1), predict what that user might rate a movie, or perform other tasks.

Figure 2: Movie Lens uses collaborative filtering to predict that this user is likely to rate the movie "Holes" 4 out of 5 stars.

To be more formal, a rating consists of the association of two things – user and item.

CF systems determine the quality of items. Items can consist of anything for which a human can provide a rating, such as art, books, CDs, journal articles, or vacation destinations.

Ratings in a collaborative filtering system can take on a variety of forms.

- Scalar ratings can consist of either numerical ratings, such as the 1-5 stars provided in MovieLens or ordinal ratings such as strongly agree, agree, neutral, disagree, strongly disagree.

- Binary ratings model choices between agree/disagree or good/bad.

• Unary ratings can indicate that a user has observed or purchased an item, or otherwise rated the item positively. The absence of a rating indicates that we have no information relating the user to the item (perhaps they purchased the item somewhere else).

Ratings may be gathered through explicit means, implicit means, or both.

b. *Uses*

This section explores user tasks that CF supports, then the services that CF systems provide.

User Tasks

Tasks for which people use collaborative filtering include

1. **Help me find new items I might like.** In a world of information overload, I cannot evaluate all things. Present a few for me to choose from. This has been applied most commonly to consumer items (music, books, movies), but may also be applied to research papers, web pages, or other ratable items.

2. **Advise me on a particular item.** I have a particular item in mind; does the community know whether it is good or bad?

3. **Help me find a user (or some users) I might like**. Sometimes, knowing who to focus on is as important as knowing what to focus on. This might help with forming discussion groups , matchmaking, or connecting users so that they can exchange recommendations socially.

4. **Help our group find something new that we might like.** CF can help groups of people find items that maximize value to group as a whole. For example, a couple that wishes to see a movie together or a research group that wishes to read an appropriate paper.

5. **Domain-specific tasks**. For example, a research paper recommender might also wish to support

    a) Recommend papers that this paper should cite.

    b) Recommend papers that should cite this paper.

Collaborative Filtering System Functionality

There are also broad abstract families of tasks that CF systems support. Here are the broad families of common CF system functionality:

1. **Recommend items.** Show a list of items to a user, in order of how useful they might be. Often this is described as predicting what the user would rate the item, then ranking the items by this predicted rating.

2. **Predict for a given item.** Given a particular item, calculate its predicted rating. Note that prediction can be more demanding than recommendation. To recommend items, a system only needs to be prepared to offer a few alternatives, but not all. To provide predictions for a particular item, a system must be prepared to say something about any requested item, even rarely rated ones.

3. **Constrained recommendations: Recommend from a set of items.** Given a particular set or a constraint that gives a set of items, recommend from within that set. For example:

   "Consider the following scenario. Mary's 8-year-old nephew is visiting for the weekend, and she would like to take him to the movies. She would like a comedy or family movie rated no "higher" than PG-13. She would prefer that the movie contain no sex, violence or offensive language, last less than two hours and, if possible, show at a theater in her neighborhood. Finally, she would like to select a movie that she herself might enjoy."

Suitable domains for collaborative filtering

One might simply take a user application, implement it with a CF system, and hope it will work. However, CF is better known to be effective in domains with certain properties. These properties are grouped below into data distribution, underlying meaning, and data persistence.

*Data distribution***.** These properties are about the numbers and shape of the data:

1. There are many items. If there are only a few items to choose from, the user can learn about them all without need for computer support.

2. There are many ratings per item. If there are only a few ratings per item, there may not be enough information to provide useful predictions or recommendations.

3. **There are more users rating than items to be recommended.** A corollary of the previous paragraph is that often you'll need more users than the number of items that you want to be able to capably recommend. More precisely, if there are few ratings per user, you'll need many users. Lots of systems are like this. As another example, with one million users, a CF system might be able to make recommendations for a hundred thousand items, but may only be able to make confident predictions for ten thousand or fewer, depending on the distribution of ratings across items. The ratings distribution is almost always very skewed: a few items get most of the ratings, a long tail of items that get few ratings. Items in this long tail will not be confidently predictable.

4. **Users rate multiple items.** If a user rates only a single item, this provides some information for summary statistics, but no information for relating the items to each other.

Underlying meaning. These properties are of the underlying meaning of the data:

1. **For each user of the community, there are other users with common needs or tastes.** CF works because people have needs or tastes in common. If a person has tastes so unique that they are not shared by anybody else, then CF cannot provide any value.

2. **Items are homogenous.** That is to say, by all objective consumption criteria they are similar, and they differ only in subjective criteria. Music albums are like this. Most are similarly priced, similar to buy, of a similar length. Books or research papers are also like this. Items sold at a department store are not like this: some are cheap, some very expensive. For example, if you buy a hammer, perhaps you should not be recommended a refrigerator.

Data persistence. These are properties of how long the data is relevant:

1. **Items persist.** Not only does a CF system need a single item to be rated by many people, but also requires that people share multiple rated items – that there is overlap in the items they rate. All of this means that if items are only important for a short time, these requirements are hard to meet. For example, news stories: many appear per day, and many probably are only interesting for a few days.

2. **Taste persists.** CF has been most successful in domains where users' tastes don't change rapidly: e.g., movies, books, and consumer electronics. If tastes change frequently or rapidly, then older ratings may be less useful. An example might be clothing, where someone's taste from five years ago may not be relevant.

c. *Collaborative Filtering Algorithms: Theory and Practice*

Collaborative Filtering (CF) systems work by collecting user feedback in the form of ratings for items in a given domain and exploiting similarities in rating behavior amongst several users in determining how to recommend an item. CF methods can be further sub-divided into ***neighborhood-based*** and ***model-based*** approaches. Neighborhood-based methods are also commonly referred to as ***memory based*** approaches [10].

Neighborhood-based Collaborative Filtering/Memory Based Approach

A. *User Based Collaborative filtering*

In neighborhood-based techniques, a subset of users is chosen based on their similarity to the active user, and a weighted combination of their ratings is used to produce predictions for this user. Most of these approaches can be generalized by the algorithm summarized in the following steps:

1. Assign a weight to all users with respect to similarity with the active user.

2. Select k users that have the highest similarity with the active user – commonly called the ***neighborhood.***

3. Compute a prediction from a weighted combination of the selected neighbors' ratings.

In step 1, the weight $w_{a,u}$ is a measure of similarity between the user u and the active user a. The most commonly used measure of similarity is the Pearson correlation coefficient between the ratings of the two users, defined below:

$$w_{a,u} = \frac{\sum_{i \in I} (r_{a,i} - \bar{r}_a)(r_{u,i} - \bar{r}_u)}{\sqrt{\sum_{i \in I} (r_{a,i} - \bar{r}_a)^2 \sum_{i \in I} (r_{u,i} - \bar{r}_u)^2}}$$

where I is the set of items rated by both users, $r_{u,i}$ is the rating given to item i by user u, and $\overline{r_u}$ is the mean rating given by user u.

In step 3, predictions are generally computed as the weighted average of deviations from the neighbor's mean, as in:

$$p_{a,i} = \overline{r}_a + \frac{\sum_{u \in K} (r_{u,i} - \overline{r}_u) \times w_{a,u}}{\sum_{u \in K} w_{a,u}}$$

where $p_{a,i}$ is the prediction for the active user a for item i, $w_{a,u}$ is the similarity between users a and u, and K is the neighborhood or set of most similar users.

Similarity based on Pearson correlation measures the extent to which there is a linear dependence between two variables. Alternatively, one can treat the ratings of two users as a vector in an m-dimensional space, and compute similarity based on the cosine of the angle between them, given by:

$$w_{a,u} = \cos(\vec{r}_a, \vec{r}_u) = \frac{\vec{r}_a \cdot \vec{r}_u}{\|\vec{r}_a\|_2 \times \|\vec{r}_u\|_2} = \frac{\sum_{i=1}^{m} r_{a,i} r_{u,i}}{\sqrt{\sum_{i=1}^{m} r_{a,i}^2} \sqrt{\sum_{i=1}^{m} r_{u,i}^2}}$$

When computing cosine similarity, one cannot have negative ratings, and unrated items are treated as having a rating of zero. Empirical studies have found that Pearson correlation generally performs better [10].

Practical challenges of user based algorithms

The user-based nearest neighbor algorithm captures how word-of-mouth recommendation sharing works and it can detect complex patterns given enough users; however it has practical challenges.

Ratings data is often sparse and pairs of users with few co-ratings are prone to skewed relations. For example, if users share only three co rated items, it is not uncommon for the ratings to match

almost exactly (a similarity score of 1).if such similarities are not adjusted , these skewed neighbors can dominate a user's neighborhood.

The original user-based algorithm as implemented in GroupLens included all users in a CF system in a prediction neighborhood. Later algorithms improved accuracy and efficiency by limiting the prediction calculation to a user's closest k neighbors[11].

B. *Item-based Collaborative Filtering*: When applied to millions of users and items, conventional neighborhood-based CF algorithms do not scale well, because of the computational complexity of the search for similar users. As an alternative, Linden [9] proposed *item-to-item* Collaborative Filtering where rather than matching similar users, they match a user's rated items to similar items. In practice, this approach leads to faster online systems, and often results in improved recommendations.

In this approach similarities between pairs of items i and j are computed offline using Pearson correlation, given by:

$$w_{i,j} = \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_i)(r_{u,j} - \bar{r}_j)}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{r}_i)^2} \sqrt{\sum_{u \in U} (r_{u,j} - \bar{r}_j)^2}}$$

where U is the set of all users who have rated both items i and j, $r_{u,i}$ is the rating of user u on item i, and $\bar{r}_i$ is the average rating of the ith item across users.

Now, the rating for item i for user a can be predicted using a simple weighted average, as in:

$$p_{a,i} = \frac{\sum_{j \in K} r_{a,j} w_{i,j}}{\sum_{j \in K} |w_{i,j}|}$$

where K is the neighborhood set of the k items rated by a that are most similar to i.

Practical Challenges in Item-based Algorithms

Theoretically, the size of the model could be as large as the square of the number of items. In practice, this size can be reduced by only storing correlations for item pairs with more than k

coratings. The model can be pruned even further by only retaining the top n correlations for each item. Such modifications yield item-based algorithms that are relatively efficient in both memory usage and CPU performance. Pruning many of the correlations means that it may be more difficult to make a prediction for a given target item and user, since the items correlated with the user's ratings may not contain the target item.

As in the user algorithm, item pairs with few coratings can lead to skewed correlations and care must be exercised to not let skewed correlations dominate a prediction.

Below are several extensions to neighborhood-based CF, which have led to improved performance.

Significance Weighting: It is common for the active user to have highly correlated neighbors that are based on very few co-rated (overlapping) items. These neighbors based on a small number of overlapping items tend to be bad predictors. One approach to tackle this problem is to multiply the similarity weight by a *Significance Weighting* factor, which devalues the correlations based on few corated items [11].

Default Voting: An alternative approach to dealing with correlations based on very few co-rated items, is to assume a default value for the rating for items that have not been explicitly rated. In this way we can now compute correlation using the union of items rated by users being matched ($I_a \cap I_u$), as opposed to the intersection. Such a *default voting* strategy improves Collaborative Filtering .

Case Amplification: In order to favor users with high similarity to the active user, *case amplification* was introduced which transforms the original weights to

$$w'_{a,u} = w_{a,u} \cdot |w_{a,u}|^{\rho-1}$$

where ρ is the amplification factor, and $\rho \geq 1$.

Model Based Approaches

Model-based algorithms use the collection of ratings to learn a model, which is then used to make rating predictions. Some of the commonly used techniques for the model based methods

are Artificial Neural Networks, Bayesian Networks, Clustering, Linear Regression, Probablistic Models.

d. *Characteristics and Challenges of Collaborative Filtering*

E-commerce recommendation algorithms often operate in a challenging environment, especially for large online shopping companies like *eBay* and *Amazon*. Usually, a recommender system providing fast and accurate recommendations will attract the interest of customers and bring benefits to companies. For *CF* systems, producing high quality predictions or recommendations depends on how well they address the challenges, which are characteristics of CF tasks as well.

Data Sparsity. In practice, many commercial recommender systems are used to evaluate very large product sets. The user-item matrix used for collaborative filtering will thus be extremely sparse and the performances of the predictions or recommendations of the *CF* systems are challenged.

The data sparsity challenge appears in several situations, specifically, the ***cold start*** [1] problem occurs when a new user or item has just entered the system, it is difficult to find similar ones because there is not enough information. New items cannot be recommended until some users rate it, and new users are unlikely given good recommendations because of the lack of their rating or purchase history. ***Coverage*** can be defined as the percentage of items that the algorithm could provide recommendations for. The *reduced coverage* problem occurs when the number of users' ratings may be very small compared with the large number of items in the system, and the recommender system may be unable to generate recommendations for them. ***Neighbor transitivity*** refers to a problem with sparse databases, in which users with similar tastes may not be identified as such if they have not both rated any of the same items. This could reduce the effectiveness of a recommendation system which relies on comparing users in pairs and therefore generating predictions.

Scalability. When numbers of existing users and items grow tremendously, traditional *CF* algorithms will suffer serious scalability problems, with computational resources going beyond practical or acceptable levels. For example, with tens of millions of customers (M) and millions of distinct catalog items (N), a *CF* algorithm with the complexity of $O(n)$ is already too large. As well, many systems need to react immediately to online requirements and make

recommendations for all users regardless of their purchases and ratings history, which demands a high scalability of a *CF* system .

Synonymy. Synonymy refers to the tendency of a number of the same or very similar items to have different names or entries. Most recommender systems are unable to discover this latent association and thus treat these products differently. For example, the seemingly different items "*children movie*" and "*children film*" are actual the same item, but memory-based *CF* systems would find no match between them to compute similarity. Indeed, the degree of variability in descriptive term usage is greater than commonly suspected. The prevalence of synonyms decreases the recommendation performance of *CF* systems.

Shilling Attacks. In cases where anyone can provide recommendations, people may give tons of positive recommendations for their own materials and negative recommendations for their competitors. It is desirable for *CF* systems to introduce precautions that discourage this kind of phenomenon.

Other Challenges. As people may not want their habits or views widely known, *CF* systems also raise concerns about personal privacy. Increased noise (or sabotage) is another challenge, as the user population becomes more diverse.

2.2.3 Demographic Filtering Systems

A general technique people use to build models of other people very quickly is the evocation of stereotypes or clusters of characteristics. A stereotype is a collection of frequently occurring characteristics of users. Demographic recommender systems makes use of descriptions of people to learn the relationships between a single item and the class or type of people who liked it. Demographic based recommender systems use prior knowledge on demographic information about the users and their opinions for the recommended items as basis for recommendations. Demographic systems are stereotypical, because they depend on the assumption that all users belonging to a certain demographic group have similar taste or preference.

A Demographic recommender system, Grundy [12], was one of the first book recommender system developed. Grundy bunds models of its users, with the aid of stereotypes, and then

exploits those models to guide it in its task, suggesting novels that people may find interesting. Users enter keywords describing their personality, not their information need in order to create a user profile. Grundy then associates terms used in the users' self-descriptions with pre-defined stereotypes. These stereotypes expand into attribute rating pairs describing the users' information needs that are aggregated to form the object ratings when making predictions.

Demographic techniques attempt to form "people-to-people" correlations like collaborative filtering, but uses different data.

Demographic filtering systems have the following **advantages**:

1. The advantage of a demographic approach is that it does not require a history of user ratings of the type required by collaborative and content based techniques.

2. Demographic approach is quick and straightforward method for making assumptions based on limited observations.

Demographic filtering systems have the following **disadvantages**:

1. For the demographic filtering to be effective, it is necessary to collect complete demographic information about users. In practice, it is difficult to collect such information as it involves privacy issues.

2. Demographic filtering systems suffer from both new-user cold-start problem and new-item cold start problem.

3. The formation of demographic clusters is based on a generalization of the user's interests. So, demographic systems try to recommend the same item to people with similar demographic profiles and the recommendations are too general.

2.2.4 Hybrid Recommender Systems

Several recommendation systems use a hybrid approach by combining collaborative and content-based methods, which helps to avoid certain limitations of content-based and collaborative systems.

Different ways to combine collaborative and content-based methods into a hybrid recommender system can be classified as follows:

1. implementing collaborative and content - based methods separately and combining their predictions,

2. incorporating some content-based characteristics into a collaborative approach,

3. incorporating some collaborative characteristics into a content-based approach,

4. constructing a general unifying model that incorporates both content-based and collaborative characteristics.

2.3 Evaluation Metrics

The quality of a recommender system can be decided on the result of evaluation. The type of metrics used depends on the type of *CF* applications.

2.3.1 Accuracy

The most prominent evaluation metrics in the research literature measure the accuracy of the system's predictions. Accuracy can either be measured as the magnitude of error between the predicted rating and the true rating, or the magnitude of error between the predicted ranking and the "true" ranking.

According to Herlocker[13], metrics evaluating recommendation systems can be broadly classified into the following broad categories:

predictive accuracy metrics, such as Mean Absolute Error (*MAE*) and its variations;

classification accuracy metrics, such as *precision*, *recall*, *F1-measure*, and *ROC sensitivity*;

rank accuracy metrics, such as *Pearson's product-moment correlatio*n, *Kendall's Tau*, *Mean Average Precision (MAP)*, *half-life utility*, and *normalized distance-based performance metric* (*NDPM*) .

The commonly-used *CF* metrics -*MAE*, *NMAE*, *RMSE.*

Mean Absolute Error (MAE) and Normalized Mean Absolute Error (NMAE). Instead of *classification accuracy* or *classification error*, the most widely used metric in *CF* research literature is *Mean Absolute Error (MAE)*, which computes the average of the absolute difference between the predictions and true ratings

$$MAE = \frac{\sum_{(i,j)} \left| p_{i,j} - r_{i,j} \right|}{n},$$

where n is the total number of ratings over all users, $p_{i,j}$ the predicted rating for user i on item j, and $r_{i,j}$ is the actual rating. The lower the *MAE*, the better the prediction.

Different recommender systems may use different numerical rating scales.

***Normalized Mean Absolute Error* (*NMAE*)** normalizes *MAE* to express errors as percentages of full scale:

$$NMAE = \frac{MAE}{r_{\max} - r_{\min}},$$

where r_{max} and r_{min} are the upper and lower bounds of the ratings.

Root Mean Squared Error (RMSE).

Root Mean Squared Error (RMSE) is becoming popular partly because it is

The *Netflix prize* metric for movie recommendation performance:

$$RMSE = \sqrt{\frac{1}{n} \sum_{(i,j)} \left(p_{i,j} - r_{i,j} \right)^2},$$

where n is the total number of ratings over all users, $p_{i,j}$ is the predicted rating for user i on item j, and $r_{i,j}$ is the actual rating again. *RMSE* amplifies the contributions of the absolute errors between the predictions and the true values.

Although accuracy metrics have greatly helped the field of recommender systems, the recommendations that are most accurate are sometimes not the ones that are most useful to users, for example, users might prefer to be recommended with items that are unfamiliar with them, rather than the old favorites they do not likely want again. We therefore need to explore other evaluation metrics.

2.4 State of Art

Although the roots of recommender systems can be traced back to the extensive work in cognitive science, approximation theory, information retrieval, forecasting theories, and also have links to management science and to consumer choice modeling in marketing, recommender systems emerged as an independent research area in the mid-1990s when researchers started focusing on recommendation problems that explicitly rely on the ratings structure. There has been much work done both in the industry and academia on developing new approaches to recommender systems over the last decade. The interest in this area still remains high because it constitutes a problem-rich research area and because of the abundance of practical applications that help users to deal with information overload and provides personalized recommendations, content, and services to them.
What constitutes a recommendation problem has already been explained.
Though each of the types of recommender systems has also been explained, we present the major work in each of the types with the help of Table 3.

Recommendation techniques are grouped into heuristic based and model based. We describe the major innovations with respect to each of the type.

Content based:

The content-based approach to recommendation has its roots in information retrieval and information filtering research. Because of the significant and early advancements made by the information retrieval and filtering communities and because of the importance of several text-based applications, many current content-based systems focus on recommending items containing textual information, such as documents, Web sites (URLs), and Usenet news

messages. The improvement over the traditional information retrieval approaches comes from the use of user profiles that contain information about users' tastes, preferences, and needs. The profiling information can be elicited from users explicitly, e.g., through questionnaires, or implicitly—learned from their transactional behavior over time.

More formally, let Content(s) be an item profile, i.e., a set of attributes characterizing item s. It is usually computed by extracting a set of features from items (its content) and is used to determine the appropriateness of the item for recommendation purposes. Since, as mentioned earlier, content-based systems are designed mostly to recommend text-based items, the content in these systems is usually described with keywords. For example, a content-based component of the Fab system [8], which recommends Web pages to users, represents Web page content with the 100 most important words. The "importance" (or " informativeness ") of word k_j in document d_j is determined with some weighting measure w_{ij} that can be defined in several different ways.

One of the best-known measures for specifying keyword weights in Information Retrieval is the term frequency/inverse document frequency (TF-IDF) measure that is defined as follows: Assume that N is the total number of documents that can be recommended to users and that keyword k_j appears in n_i of them. Moreover, assume that $f_{i,j}$ is the number of times keyword k_i appears in document d_j. Then, $TF_{i,j}$, the term frequency (or normalized frequency) of keyword k_i in document d_j, is defined as

$$TF_{i,j} = \frac{f_{i,j}}{\max_z f_{z,j}},$$

where the maximum is computed over the frequencies $f_{z,j}$ of all keywords k_z that appear in the document d_j. However, keywords that appear in many documents are not useful in distinguishing between a relevant document and a non relevant one. Therefore, the measure of inverse document frequency (IDF_i) is often used in combination with simple term frequency ($TF_{i,j}$) The inverse document frequency for keyword k_i is usually defined as

$$IDF_i = \log \frac{N}{n_i}.$$

Then, the TF-IDF weight for keyword k_i in document d_j is defined as

$W_{i,j} = TF_{i,j} \times IDF_i$

and the content of document d_j is defined as

Content(d_j) = (w_{1j},…….,w_{kj}).

Besides the traditional heuristics that are based mostly on information retrieval methods, other techniques for content-based recommendation have also been used, such as Bayesian classifiers and various machine learning techniques, including clustering, decision trees, and artificial neural networks. These techniques differ from information retrieval-based approaches in that they calculate utility predictions based not on a heuristic formula, such as a cosine similarity measure, but rather are based on a model learned from the underlying data using statistical learning and machine learning techniques.

Collaborative filtering:

There have been many collaborative systems developed in the academia and the industry. It can be argued that the Grundy system was the first recommender system which proposed using stereotypes as a mechanism for building models of users based on a limited amount of information on each individual user. Later on, the Tapestry system relied on each user to identify like-minded users manually. GroupLens, Video Recommender, and Ringo were the first systems to use collaborative filtering algorithms to automate prediction. Algorithms for collaborative recommendations can be grouped into two general classes: memory-based (or heuristic-based) and model-based. Memory-based algorithms essentially are heuristics that make rating predictions based on the entire collection of previously rated items by the users. Various approaches have been used to compute the similarity between users in collaborative recommender systems. In most of these approaches, the similarity between two users is based on their ratings of items that both users have rated. The two most popular approaches are correlation and cosine-based which have been explained earlier. Many performance-improving modifications, such as default voting, inverse user frequency, case amplification, and weighted-majority prediction, have been proposed as extensions to these standard correlation-based and cosine-based techniques. For example, the default voting is an extension to the memory-based approaches described above. It was observed that, whenever there are relatively few user-specified ratings, these methods would not work well in computing the similarity between users

x and y since the similarity measure is based on the intersection of the itemsets, i.e., sets of items rated by both users x and y. It was empirically shown that the rating prediction accuracy could improve if we assume some default rating value for the missing ratings .In contrast to memory-based methods, model-based algorithms use the collection of ratings to learn a model, which is then used to make rating predictions. For example, [15] proposes a probabilistic approach to collaborative filtering, where the unknown ratings are calculated as

$$r_{c,s} = E(r_{c,s}) = \sum_{i=0}^{n} i \times \Pr(r_{c,s} = i | r_{c,s'}, s' \in S_c)$$

and it is assumed that rating values are integers between 0 and n and the probability expression is the probability that user c will give a particular rating to item s given that user's ratings of the previously rated items. To estimate this probability, [15] proposes two alternative probabilistic models: cluster models and Bayesian networks. In the first model, like-minded users are clustered into classes. Given the user's class membership, the user ratings are assumed to be independent, i.e., the model structure is that of a naïve Bayesian model. The number of classes and the parameters of the model are learned from the data. The second model represents each item in the domain as a node in a Bayesian network, where the states of each node correspond to the possible rating values for each item. Both the structure of the network and the conditional probabilities are learned from the data. One limitation of this approach is that each user can be clustered into a single cluster, whereas some recommendation applications may benefit from the ability to cluster users into several categories at once.

Moreover, [16] proposed a collaborative filtering method in a machine learning framework, where various machine learning techniques (such as artificial neural networks) coupled with feature extraction techniques (such as singular value decomposition—an algebraic technique for reducing dimensionality of matrices) can be used. There have been several other model-based collaborative recommendation approaches proposed in the literature. Other collaborative filtering methods include a Bayesian model, a probabilistic relational model, a linear regression , and a maximum entropy model .

Hybrid:

Hybrid recommendation approaches were used to avoid limitations of earlier approaches. One way to build hybrid recommender systems is to implement separate collaborative and content-based systems. Two different scenarios could be there. First, we can combine the outputs (ratings) obtained from individual recommender systems into one final recommendation using either a linear combination of ratings or a voting scheme . Alternatively, we can use one of the individual recommenders, at any given moment choosing to use the one that is "better" than others based on some recommendation "quality" metric. Another most popular approach is to use some dimensionality reduction technique on a group of content based profiles.

Popescul et al. and Schein et al. [17] proposed a unified probabilistic method for combining collaborative and content-based recommendations, which is based on the probabilistic latent semantic analysis.

Recommendation Approach	Recommendation Technique	
	Heuristic-based	Model-based
Content-based	Commonly used techniques: • TF-IDF (information retrieval) • Clustering Representative research examples: • Lang 1995 • Balabanovic & Shoham 1997 • Pazzani & Billsus 1997	Commonly used techniques: • Bayesian classifiers • Clustering • Decision trees • Artificial neural networks Representative research examples: • Pazzani & Billsus 1997 • Mooney et al. 1998 • Mooney & Roy 1999 • Billsus & Pazzani 1999, 2000 • Zhang et al. 2002
Collaborative	Commonly used techniques: • Nearest neighbor (cosine, correlation) • Clustering • Graph theory Representative research examples: • Resnick et al. 1994 • Hill et al. 1995 • Shardanand & Maes 1995 • Breese et al. 1998 • Nakamura & Abe 1998 • Aggarwal et al. 1999 • Delgado & Ishii 1999 • Pennock & Horwitz 1999 • Sarwar et al. 2001	Commonly used techniques: • Bayesian networks • Clustering • Artificial neural networks • Linear regression • Probablistic models Representative research examples: • Billsus & Pazzani 1998 • Breese et al. 1998 • Ungar & Foster 1998 • Chien & George 1999 • Getoor & Sahami 1999 • Pennock & Horwitz 1999 • Goldberg et al. 2001 • Kumar et al. 2001 • Pavlov & Pennock 2002 • Shani et al. 2002 • Yu et al. 2002, 2004 • Hofmann 2003, 2004 • Marlin 2003 • Si & Jin 2003
Hybrid	Combining content-based and collaborative components using: • Linear combination of predicted ratings • Various voting schemes • Incorporating one component as a part of the heuristic for the other Representative research examples: • Balabanovic & Shoham 1997 • Claypool et al. 1999 • Good et al. 1999 • Pazzani 1999 • Billsus & Pazzani 2000 • Tran & Cohen 2000 • Melville et al. 2002	Combining content-based and collaborative components by: • Incorporating one component as a part of the model for the other • Building one unifying model Representative research examples: • Basu et al. 1998 • Condliff et al. 1999 • Soboroff & Nicholas 1999 • Ansari et al. 2000 • Popescul et al. 2001 • Schein et al. 2002

Table 2: Classification of recommender systems research

2.5 Summary

This chapter explains Web 2.0, its various components, technologies used, how recommender systems are related to it, three basic types of recommender system-content, collaborative and hybrid along with their limitations and advantages. The next section of this chapter explains the various research techniques that have been adopted down the years in this field.

CHAPTER 3

RESEARCH METHODOLOGY

CHAPTER 3

RESEARCH METHODOLOGY

The entire research framework is divided into two sections

1. How to deal with sparseness issue of dataset?
2. How to improve scalability?

The terms Sparsity and Scalability is being elaborated with reference to a recommender system.

Sparsity : Commercial recommender systems in general are used to evaluate very large product sets. In a user – item rating database, though users are very active, there are a few rating of the total number of items available. The user-item matrix is thus extremely sparse. Since a collaborative filtering algorithm is mainly based on similarity measures computed over the co-rated set of items, thus large levels of sparsity can lead to less accuracy and can challenge the predictions or recommendations of the *CF* systems.

Scalability: A Collaborative filtering algorithm is assumed to be efficient if it is able to filter items that are interesting to users. But, they require computations that are very expensive and grow non-linearly with the number of users and items in a database. In general, the whole ratings database is searched in collaborative filtering and thus it suffers from poor scalability when more and more users and items are added into the database.

This chapter covers techniques adopted for dealing with the above issues. Two techniques **Weighted Slope One Scheme** and **Item Classification** have been used to determine vacant ratings in the given sparse data set. Further to deal with scalability issue **K-means Clustering** have been used.

We have compared two collaborative filtering algorithm results in the next chapter.In each of the algorithm, different techniques have been used to predict vacant ratings mentioned above (weighted slope one scheme and item classification). K-means clustering is then used further to group items in both cases for producing the final recommendations.

3.1 Sparsity reduction (Pre – prediction)

For reducing the sparseness of data, the techniques adopted are:

3.1.1 Weighted slope one scheme

Slope One is a family of algorithms used for collaborative filtering. It was introduced in a 2005 paper by **Daniel Lemire** and **Anna Maclachlan**. These are very simple to implement and their accuracy is often on par with more complicated and computationally expensive algorithms .The slope one predictors was first introduced for online rating. The basic idea is to answer the question how a user would rate a give item, given other users' ratings[17]. The following characteristics are desirable in a recommender system:

• ***easy to implement and maintain***: all aggregated data should be easily interpreted by the average engineer;

• ***updateable on the fly***: the schemes should not rely exclusively on batch processing;

• ***not demanding from new users***: as long as one can guess how a use feels about one item, it should immediately be able to offer recommendations;

• ***efficient at query time***: speed should not depend on the number of users and extra storage could be used to precompute the queries;

• ***reasonably accurate***: the schemes should be competitive with more expensive or complex schemes even though simplicity and convenience is primarily desirable.

The Weighted Slope One algorithm meets all these requirements. Other algorithms such as memory-based ones do not allow us to precompute the recommendations and require that the user has rated several items before a sensible recommendation can be made.

Slope One algorithms work on the intuitive principle of a "**popularity differential**" between items for users. In a pair wise fashion, we determine how much better one item is liked than another. One way to measure this differential is simply to subtract the average rating of the two items. In turn, this difference can be used to predict another user's rating of one of those items, given their rating of the other.

The slope one method uses a simpler form of regression f(x) = x + b, hence the name "slope one". The free parameter is simply the average difference between the two items' ratings.

Example:

Consider the following table.

	Item 1	**Item 2**	**Item 3**
John	5	3	2
Mark	3	4	Didn't rate it
Lucy	Didn't rate it	2	5

Table 3-1: Sample Rating Database

In this case, the average difference in ratings between item 2 and 1 is (2 + (-1)) / 2=0.5. Hence, on average, item 1 is rated above item 2 by 0.5. Similarly, the average difference between item 3 and 1 is 3. Hence, if we predict the rating of Lucy for item 1 using her rating for item 2, we get 2 + 0.5 = 2.5. Similarly, if we try to predict her rating for item 1 using her rating of item 3, we get 5 + 3=8.

If a user rated several items, the predictions are simply combined using a weighted average where a good choice for the weight is the number of users having rated both items. In the above example, we would predict the following rating for Lucy on item 1:

$(2 \times 2.5 + 1 \times 8)/(2 + 1) = 13/3 = 4.33$

Hence, given n items, to implement Slope One, all that is needed is to compute and store the average differences and the number of common ratings for each of the n^2 pairs of items[16].

Notation : The following notation in used for describing the scheme. The ratings from a given user, is called an *evaluation*, is represented as an incomplete array u, where u_i is the rating that this user gives to item i. The subset of the set of items consisting of all those items which are rated in u is $S(u)$. The set of all evaluations in the training set is χ. The number of elements in a set S is *card* (S). The average of ratings in an evaluation u is denoted u. The set $S_i(\chi)$ is the set of all evaluations $u \in \chi$ such that they contain item i ($i \in S(u)$). Given two evaluations u, v, we define the scalar product $\langle u,v \rangle$ as $\Sigma_{i \in S(u) \cap S(v)} u_i v_i$. Predictions, $P(u)$, represent a vector where each component is the prediction corresponding to one item: predictions depend implicitly on the training set χ [15].

Expressing Slope one scheme using above notation

Formally, given two evaluation arrays v_i and w_i with $i = 1, \ldots, n$, we search for the best predictor of the form $f(x) = x + b$ to predict w from v by minimizing $\Sigma_i (v_i + b - w_i)^2$. Deriving with respect to b and setting the derivative to zero, we get $b = (\Sigma_i wi - vi)/n$. In other words, the constant b must be chosen to be the average difference between the two arrays.

This result motivates the following scheme. Given a training set c, and any two items j and i with ratings u_j and u_i respectively in some user evaluation u (annotated as $u \varepsilon S_{j,i}(\chi)$), we consider the average deviation of item i with respect to item j as:

$$\mathrm{dev}_{j,i} = \sum_{u \in S_{j,i}(\chi)} \frac{u_j - u_i}{card(S_{j,i}(\chi))}.$$

The symmetric matrix defined by $\mathrm{dev}_{j,i}$ can be computed once and updated quickly when new data is entered. Given that $\mathrm{dev}_{j,i} + u_i$ is a prediction for u_j given u_i, a reasonable predictor might be the average of all such predictions

$$P(u)_j = \frac{1}{card(R_j)} \sum_{i \in R_j} (\mathrm{dev}_{j,i} + u_i)$$

where $R_j = \{i | i \ \varepsilon \ S(u), i \neq j, card(S_{j,i}(\chi)) > 0\}$ is the set of all relevant items.

One of the drawbacks of SLOPE ONE is that the number of ratings observed is not taken into consideration. Intuitively, to predict user *A*'s rating of item *L* given user *A*'s rating of items *J* and *K*, if 2000 users rated the pair of items *J* and *L* whereas only 20 users rated the pair of items *K* and *L*, then user *A*'s rating of item *J* is likely to be a far better predictor for item *L* than user *A*'s rating of item *K* is. Thus, we define the WEIGHTED SLOPE ONE prediction as the following weighted average

$$P^{wS1}(u)_j = \frac{\sum_{i \in S(u)-\{j\}} (\mathrm{dev}_{j,i} + u_i) c_{j,i}}{\sum_{i \in S(u)-\{j\}} c_{j,i}}$$

Where $c_{j,i} = card\,(S_{j,i}\ (\chi))$

3.1.2 Item Classification

This approach classifies the items to pre-produce the ratings, where necessary.

Item Attribute Content

Items may be categorized or clustered based on the attributes of the items. For Example, in the context of movies, every movie can be classified according to the "genre" attribute of each item. In our work , we have used movies as the items and the various genre that they could belong are Action, Adventure , Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western. So in all we have divided the collection of all movies in 18 groups. Table 3 shows examples of descriptive information of items.

item \ attribute	A1	A2	….. ……	At
Item1	r11	r12	….. ……	r1t
Item2	r21	r22	….. ……	r2t
….. ……	….. ……	….. ……	….. ……	….. ……

Itemn	rn1	rn2	….. ……	Rnt

Table 3-2 : Item-item attribute table

Where rij denotes the express value of the item to its attribute. The symbol n denotes the total number of items, and t denotes the total number of item attributes[14].

Once the items have been classified according to the attribute content, then in the sub-matrix formed user based collaborative filtering is used to fill the vacant rating. An item can belong to one or more genre. In such case the mean of the value from each of the submatrix formed is used as the value for that item.

3.2 Improving Scalability (Using Item Clustering)

Scalibility is improved using K-means clustering.In this first the items are clustered into groups. Next the target item for which recommendation needs to be computed is matched with the centroid of each of the clusters formed. The nearest similarity clusters are identified and then nearest similarity items within these clusters are identified .These neighbors' are further used to predict the value for the target item. Scalibility is better because the entire rating database is not searched rather specific clusters are searched to identify similar items.

3.2.1 K-means clustering

Item clustering techniques identify groups of items that have similar ratings. Once the clusters are created , prediction for a target item is made by averaging the opinions of other item in that cluster. Item clustering gives good performance because the size of the group that needs to be analyzed becomes much smaller.

The basic idea is to divide the items of a collaborative filtering system using item clustering algorithm and use the divide as neighborhoods as figure 1 shows

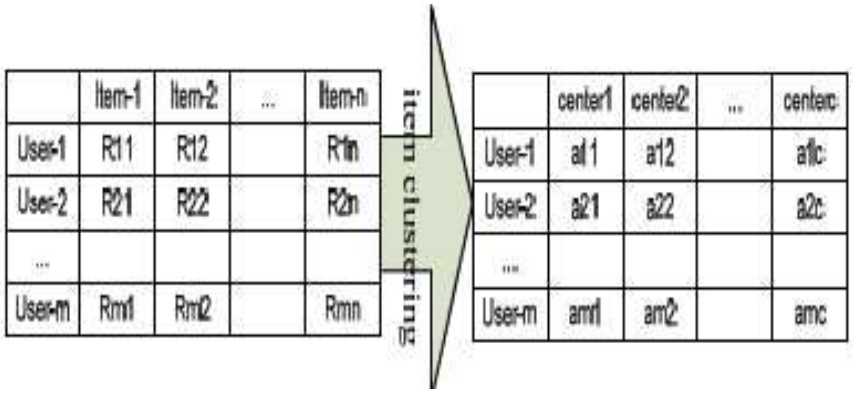

Figure 3- 1: Collaborative filtering based on item clustering

Where Rij is the rating of the user i to the item i, aij the average rating of the user i to the item center j , m is the number of all users , n is the number of all items , and c is the number of item centers.

K- means is used as the basic clustering algorithm. K specifies the number of clusters to be created.

K-means (In General)

The *k*-means algorithm assigns each point to the cluster whose center (also called centroid) is nearest. The center is the average of all the points in the cluster

Example: The data set has three dimensions and the cluster has two points: $X = (x_1,x_2,x_3)$ and $Y = (y_1,y_2,y_3)$. Then the centroid Z becomes $Z = (z_1,z_2,z_3)$, where $z_1 = (x_1 + y_1)/2$,

$Z_2 = (x_2 + y_2)/2$ and $z_3 = (x_3 + y_3)/2$

The algorithm steps are:

- Choose the number of clusters, *k*.
- Randomly generate *k* clusters and determine the cluster centers, or directly generate *k* random points as cluster centers.
- Assign each point to the nearest cluster center, where "nearest" is defined with respect to one of the distance measures (Like Euclidean distance , Manhattan distance , Mahalanobis distance etc)

- Recompute the new cluster centers.
- Repeat the two previous steps until some convergence criterion is met (usually that the assignment hasn't changed).

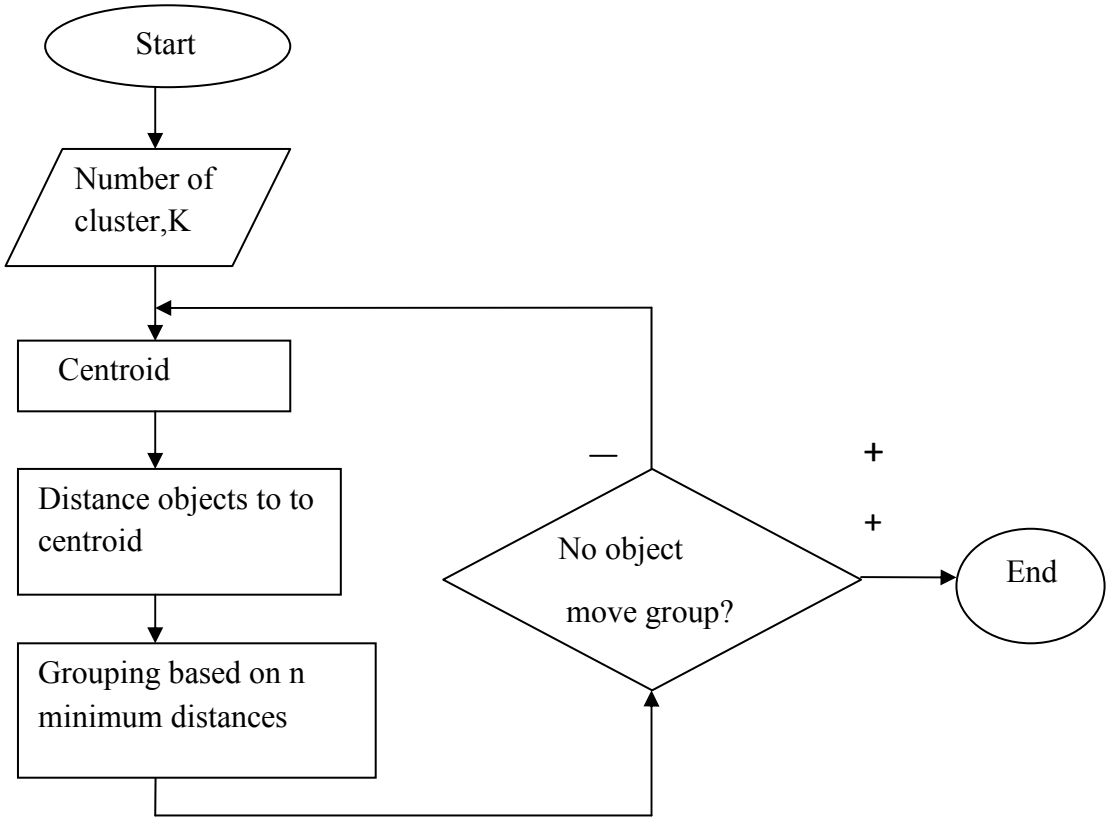

Figure 3-2 : **Flowchart for K-means clustering**

Specific Algorithm:

The algorithm first takes K items as the centers of K unique clusters. Each of the remaining items is then compared to the closest center. In the following passes, the cluster centers are re-computed based on cluster centers formed in the previous pass and the cluster membership is re-evaluated.

Input : clustering number k

Output: item-center matrix

Begin

Select user set U={U1, U2, …, Um};

```
Select item set I={I1, I2, …, In};
Choose the top k rating items as the clustering CI={CI1, CI2, …, CIk};
The k clustering center is null as c={c1, c2, …, ck};
do
    for each item Ii □ I
        for each cluster center CIj □ CI
            calculate the sim(Ii, CIj);
        end for
        sim(Ii, CIx)=max{sim(Ii, CI1), sim(Ii, CI2), …,sim(Ii, CIk)};
        cx=cx U Ii
    end for
    for each cluster ci □ c
        for each user Ij □ I
        CIi=average(ci, Ij);
        end for
    end for
while (CU and c is not change)
End
```

The pearson's correlation, as following formula, is used to measure the linear correlation between two vectors of ratings as the target item t and the remaining item r.

$$sim(t,r) = \frac{\sum_{i=1}^{m} (R_{it} - A_t)(R_{ir} - A_r)}{\sqrt{\sum_{i=1}^{m} (R_{it} - A_t)^2 \sum_{i=1}^{m} (R_{ir} - A_r)^2}}$$

Where R_{it} is the rating of the target item t by user i, R_{ir} is the rating of the remaining item r by user i, A_t is the average rating of the target item t for all the co-rated users, A_r is the average rating of the remaining item r for all the co-rated users, and m is the number of all rating users to the item t and item r.

K-means clustering used above groups items with maximum similarity.

3.3 Collaborative Filtering combining Sparsity Reduction techniques And Item clustering

We combine the sparsity reduction technique and item clustering for more scalable and accurate recommendations. The sparsity reduction technique help to determine vacant ratings in the entire data set thus providing dense data set. Item clustering using K-means is used to further cluster similar items .The scalability is improved using item clustering because similar items can be found easily by selecting nearest matching cluster centroids with the target item[18].

Once the items are clustered , the item centers are obtained. This center is represented as an average rating over all items in the cluster. The target item neighbors are chosen in some of the item center clustering. Pearson's correlation is used to compute similarity between target item and the item centers. Once the similarity is calculated between the target item and the item centers , the items in the most similar centers are chosen as the candidates.

Once the target item nearest clustering centers are chosen, then similarity is calculated between the target item and items in the selected clustering centers. The top K most similar items based on cosine measure are selected. Cosine measure looks at the angle between two vectors of ratings as the target item t and the remaining item r. The following formula is used:

$$sim(t,r) = \frac{\sum_{i=1}^{m} R_{it} R_{ir}}{\sqrt{\sum_{i=1}^{m} R_{it}^{2} \sum_{i=1}^{m} R_{ir}^{2}}}$$

Where R_{it} is the rating of the target item t by user i, R_{ir} is the rating of the remaining item r by user i, and m is the number of all rating users to the item t and item r.

Once the membership of items is computed, we calculate the weighted average of neighbors' ratings, weighted by their similarity to the target item. The rating of the target user u to the target item t is as following:

$$P_{ut} = \frac{\sum_{i=1}^{c} R_{ui} \times sim(t,i)}{\sum_{i=1}^{c} sim(t,i)}$$

Where R_{ui} is the rating of the target user u to the neighbour item i, sim(t, i) is the similarity of the target item t and the neighbour it user i for all the co-rated items, and m

is the number of all rating users to the item t and item r.

3.4 Summary

This chapter describes the sparsity reduction techniques (weighted slope one scheme and Item classification) and scalabilty improving technique (Item Clustering).Sparsity reduction technique provides dense data set. Slope One algorithm work on the principle of a "**popularity differential**" between items for users. In general, its determined how much one item is liked better over other. This information is used to predict rating for a user, given ratings of other users. Item classification technique also provides with dense data set. Items are classified using item attribute content .User based collaborative filtering is used to determine unavailable ratings, after the items are classified. K-means clustering is used to cluster items to alleviate scalibility issue. Item clustering reduces the effort of searching in the entire rating database. Rather selected clusters are searched. Once the items are clustered then cosine measure is used to select top K

most similar items in the selected clusters. Finally, the weighted average of neighbors' ratings weighted by their similarity to the target item is calculated, which is the final prediction for the target item.

CHAPTER 4

IMPLEMENTATION AND ANALYSIS

CHAPTER 4

IMPLEMENTATION AND ANALYSIS

4.1 Experimental set-up

The programming language used for source code is **Python 2.7**. Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming making it ideal language for scripting and rapid application development.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, http://www.python.org/, and may be freely distributed.

4.2 Dataset

Data is collected from Grouplens website [19].We have used Movielens collaborative filtering data set to evaluate the performance of the two algorithms. Movielens data sets were collected by the Grouplens Research Project at the university of Minnesota and Movielens is a web-based research recommender system that debuted in Fall 1997. It is an experimental data source .The data consists of 1,00,000 ratings from 943 users who rated for 1682 items(movies). The ratings are the five values of an attribute, from 1 to 5, it is the opinion that users have about movies, where 1 means the lower rating or preference and 5 represents, the maximum.Each user has registered its gender, age, occupation and zip code. The attributes about movies are: title, release date, video release date and other 19 dedicated to each possible movie gender or category (unknown, action, adventure, animation, children, comedy, crime, documentary, drama, fantasy, film-noir, horror,musical, mystery, romance, science-fiction, thriller, war and western). These last features get value 1, if the movie belongs to a specific genre and 0 otherwise. This means a movie can belong to different film genres.

4.3 Performance comparison

The item classification technique is based on the concept that the items are divided into groups based on some attribute. And then user based collaborative filtering is used within each group to determine vacant ratings. If an item belongs to more than one group, then we compute prediction for that item in each group and then aggregate the results. Traditional collaborative filtering approaches compute user similarity on a whole range of items. We believe that similarity computed as in traditional collaborative filtering is not very appropriate. The reason being that the number of items in various domains such as books or movies etc is very large. Since there are a lot of items, so the diversity between items is also very large. Users who have similar preference in one category of item may have totally different opinion for other kind of items. In our work, we have divided movies according to genre which is a kind of metadata. When we divide based on genre, we believe that items belonging to the same group will have some common characteristics. Therefore we believe user similarity computed within the group is more suitable while determining a prediction and hence gives better accuracy. The dense dataset computation was easier once the movies were divided into groups according to genre because groups creation reduced the computational efforts. On the other hand, weighted slope one algorithm is based on preference differential of items. The advantages of this technique are that it is easy to implement and maintain, efficient at query time though at expense of storage and can provide reliable recommendations even if the user is new and has given very less ratings. But this technique does not provide as much accuracy as item classification technique. We have performed item clustering on the dense data sets produced by both the techniques. But since the data set produced by item classification technique is more accurate, the item classification technique with item clustering is more preferable as compared to weighted slope one algorithm with item clustering.

4.4 Limitation

The item classification technique suffers from the following problems:

1. In some domains, there are no available metadata at all.

2. If there are various kinds of metadata available for a domain, then it becomes difficult to choose.

3. Since metadata is created by humans, it may suffer from error.

4.5 Case study

The data set from grouplens website was too huge and for dealing with sparsity, 1486126 unknown ratings need to be calculated which needed long time for computation. So we decided to implement our algorithms on a reduced data set. We consider the data set having only those items which are rated by 60 or more users. This reduced the original data set to a size having 309 users and 536 items. Total known ratings in this data set is 53536. 5% of the users are randomly selected to be the test users. From each user in the test set, ratings for 5 items were withheld and predictions were computed for those 5 items using both method1(weighted slope one scheme and item clustering) and method2 (item classification and item clustering) .For item classification technique, we need to consider the genre of movies. On reducing the data set, no movie from among the selected ones belonged to genre 'unknown'. So finally for our dataset, 18 genres were considered. The various genre that movies belonged to are Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance, Sci-Fi, Thriller, War, Western.

First the dense data set is created using each of the technique described in previous chapter to reduce sparseness. Then item clustering is done on both the dense datasets produced. K-means clustering is used to group similar items. The number of clusters created is 48. The target item for which prediction is to be done is matched with the centroid of each of the clusters formed. The neighbors in the nearest matching clusters is used to determine prediction for the target item.

Metric used to evaluate the accuracy of algorithms is MAE (Mean Absolute Error).It compares the deviation of the predicted ratings from the respective actual user ratings. The size of the neighborhood has a significant effect on the prediction quality. We varied the number of neighbors and compute the MAE. The conclusion as depicted from the Figure 4-1, which includes the Mean Absolute Errors for the two algorithms as observed in relation to the different

number of neighbors, is that Item Classification technique is better. The predicted values for each of the target item for different users(5% of total users) , along with their actual rating and absolute difference between predicted and actual ratings for each of the collaborative filtering algorithms have been shown in **Appendix C – Results**.

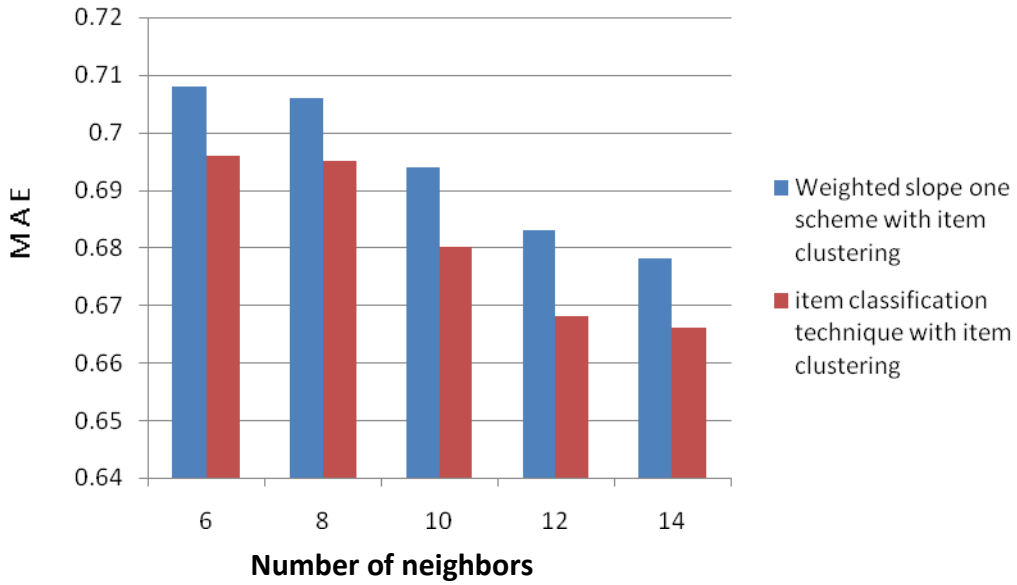

Figure 4-1: Comparing the collaborative filtering algorithms

4.6 Summary

In this chapter we have described the case study on both the collaborative algorithms using the techniques discussed in the previous chapter. We have discussed the dataset used for the algorithms along with the environment needed for implementation. We have also discussed the analysis and result of the algorithms.

CHAPTER 5

CONCLUSION

CHAPTER 5

CONCLUSION

5.1 Contributions

The term data-mining could be related to a simple analogy, finding the needle in the haystack. The Internet, nowadays, is a perennial source of large amount of informative knowledge. Web (data) mining could be partly used to solve the problems of information overload directly or indirectly. Additionally, another most promising technique for this is Web recommendation. The recommender systems have very quickly gone from the research world to popular applications. However, many problems remain to be solved. Several methods for recommender systems have emerged. However, the architectural issues of cold-start, sparse ratings, and scalability continue to dominate the field.

We have proposed and compared two collaborative filtering recommendation algorithms which helps us to alleviate the issues of sparsity and scalability as these are the dominant problems faced in the implementation of any of the recommender systems. The first algorithm joins weighted slope one scheme and item clustering technology and the other algorithm joins item classification technique and item clustering. Item clustering helps to meet the real time requirement of recommender system by reducing the search effort needed to find neighbours of the target item. We demonstrated software which can be used to make predictions based on the two collaborative filtering algorithms. We have shown a case study of these two algorithms on Movielens data set. This data set was reduced to ease computation problems. It turned out that Item classification scheme is better than weighted slope one scheme as far as accuracy is concerned. Computation time required for creating dense data set is also lesser.This collaborative filtering algorithm can be deployed in any kind of recommender systems.

The system is easier to implement and it provides us with a way to solve two dominant issues in one go.

The main problems with the proposed system:

1. It might be the case that there could exist data for which attribute identification is troublesome.
2. The proposed algorithms does not solve the cold start problem.
3. The system does not work well where the attributes have synonymous names, for eg *children movie* and *children films* refer to same thing, but the algorithm does not have any provision to shield itself from effects of synonymy.
4. The system does not provide any explanation for the predicted recommendations which is crucial for building user trust.

5.2 Future Work

The model proposed for the collaborative filtering algorithms is executed on a reduced data set from movielens. But this work can be easily extended to other datasets like Jester, datasets related to e-commerce. It will be very relevant to study its outcome on some Ecommerce related dataset because ultimately recommender systems are hugely beneficial for any online business.

We have incorporated two sparsity reduction techniques in our algorithms but there exists other sparsity reducing techniques too. Other sparsity reducing techniques are case based reasoning, content based predictor (like TAN-ELR : tree augmented naïve Bayes optimized by extended logistic regression) ,extended BMI (Bayesian Multiple Imputation) etc. which too are good .

A work could be carried out to see how well these techniques can fit in our algorithm and does they provide reasonably better results. The technique used to overcome scalability issue in our work is k-means clustering which is the simplest one. Other clustering techniques that can be tried are Fuzzy c-means clustering, clustering using genetic algorithms, hierarchical clustering etc.

It will also be worthwhile to apply significance weighting to the selected neighbors while computing the prediction for the target item and study its results which are expected to improve

the recommendations. The system could be enhanced more if it could provide explanation for the predicted value as it helps to build user trust.

We have used pearson correlation for computing similarity. But there exists better similarity measures like fractional function, exponential function which have been proposed recently. A study could also be carried out to learn how well they suit our collaborative filtering algorithm.

REFERENCES

REFERENCES

1. Goldberg, D., et. al.(1992). Using collaborative filtering to weave an information tapestry. *Communications of ACM.* 35(12), pp. 61-70.
2. Resnick, P., et.al. (1194). Grouplens: an open architecture for collaborative filtering of netnews. *Proceedings of the ACM Conference on ComputerSupported Cooperative Work,* pp. 175-186.
3. Adomavicius,Gediminas., Tuzhilin, Alexander.(2005).Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions.IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING. 17(6), pp. 734-749.
4. http://en.wikipedia.org/wiki/Recommender_system.
5. Rao, K.N., Talwar, V.G. (2008). Application Domain and Functional Classification of Recommender Systems – A Survey. *DESIDOC Journal of Library & Information Technology*. 28(3), pp. 17-35.
6. Shardanand, U., Maes, P. (1995). Social Information Filtering : Algorithms for automating 'Word of Mouth'. Proceedings of Conference Human Factors in Computing Systems.
7. Sheth, B., Maes, P.(1993). Evolving Agents for Personalized Information Filtering. Proceedings of NINTH IEEE Conference Artificial Intelligence for Applications.
8. Konstan , J.A., et. al.(1997). Grouplens: Applying Collaborative Filtering to Usenet News. Communication of ACM. 40(3), pp. 77-87.
9. Linden, G., Smith, B., York, J.(2003).Amazon.com recommendations: item-to-item collaborative filtering. IEEE Internet Computing. 7(1), pp. 76-80.
10. Breese, J.S., Heckerman, D., Kadie, C.(1998). Empirical Analysis of predictive algorithms for collaborative filtering. *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Madison.*
11. Herlocker, J.L.,et.al.(1999). An algorithmic framework for performing collaborative filtering. Proceedings of the 22nd International Conference on Research and Development in Information Retrieval, SIGIR '99.

12. Rich, E.(1979). User modeling via stereotypes. Cognitive Science, 3,pp. 329-354.
13. Herlocker, J.L.,et.al.(2004). Evaluating Collaborative Filtering Recommender Systems. ACM Transactions on Information Systems. 22(1),pp. 5-53.
14. Tan,HengSong.,Ye,HongWu .(2009).A Collaborative Filtering Recommendation Algorithm Based on Item Classification. Pacific-Asia Conference on Circuits, Communications and Systems , PACCS'09, pp.694-697.
15. Lemire, Daniel., Maclachlan, Anna .(2005). Slope One Predictors for Online Rating-Based Collaborative Filtering. SIAM Data Mining (SDM'05).
16. http://en.wikipedia.org/wiki/Slope_One.
17. Wang,Pu.,Ye,HongWu. (2009).A Personalized Recommendation Algorithm Combining Slope One Scheme and User Based Collaborative Filtering. Proceedings of the 2009 International Conference on Industrial and Information Systems, IIS '09.
18. Gong, SongJie.(2010).A Collaborative Filtering Recommendation Algorithm Based on User Clustering and Item clustering. 5(7),pp. 745-752.
19. http://www.grouplens.org.

APPENDIX A

APPENDIX A

PROGRAM CODE

To reduce original data into small data set (309 users and 536 items)

```
import math
U=[[0]*1683 for i in xrange(943)]       #matrix for storing ratings
no_users=len(U)
no_items=len(U[0])
print "no. of items=",no_items
print "no. of users=",no_users   #total no. of users
f=file("D:/preextnd.txt","r")       #file containing userid, movieid, ratings
d=[line.split() for line in f]#reading file line by line
print "length of d=",len(d)
for it in range(len(d)):
    m=int(d[it][0])
    n=int(d[it][1])
    p=int(d[it][2])
    U[m-1][n-1]=p
cc=[0]*no_items
for i in range(no_items):
    c=0
    for j in range(no_users):
        if(U[j][i]!=0):
            c=c+1
            cc[i]=c
bigcols=[]
for i in range(no_items):
    if(cc[i]>=60 ):
        bigcols.append(i)
print "bigcols=",bigcols
l=len(bigcols)
print"len of bigcols=",l
f=open('d:/itemconsder60.txt','a')
for i in range(l):
    k=bigcols[i]
    f.write(str(k+1))
    f.write('\n')
f.close()
```

Dividing data set into two files : base and test file

```
import math
U=[[0]*309 for i in xrange(536)]       #matrix for storing ratings
ti=len(U)
tu=len(U[0])
print "no. of items=",ti   #total no. of users
print "no. of users=",tu   #total no. of items
f=file("D:/slopevalnew.txt","r")#file containing userid, movieid, ratings
d=[line.split() for line in f]#reading file line by line
for it in range(len(d)):
    m=int(d[it][0])
```

```
    n=int(d[it][1])
    p=float(d[it][2])
    U[n-1][m-1]=p
remove=[5,100,262,402,515]
b=0
s=0
ni=[[0]*309 for i in xrange(531)]
held=[[0]*309 for i in xrange(5)]

for i in range(536):
       if(b<5):
        k=remove[b]
    if(k!=i ):
        for j in range(309):
            ni[s][j]=U[i][j]
        s=s+1
    else:
        b=b+1
f=open('d:/slopediv_cluster.txt','a')
for i in range(531):
    for j in range(309):
        f.write(str(i+1))
        f.write(' ')
        f.write(str(j+1))
        f.write(' ')
        f.write(str(ni[i][j]))
        f.write('\n')
f.close()
b=0
s=0
for i in range(536):
    if(b<5):
        k=remove[b]
    if(k==i):
        for j in range(309):
            held[s][j]=U[i][j]
        s=s+1
        b=b+1
f=open('d:/slopediv_test.txt','a')
for i in range(5):
    for j in range(309):
        f.write(str(i+1))
        f.write(' ')
        f.write(str(j+1))
        f.write(' ')
        f.write(str(held[i][j]))
        f.write('\n')
f.close()
```

Item Classification Technique

```
import math
U=[[0]*536 for i in xrange(309)]       #matrix for storing ratings
userratedi=[]
tu=len(U)
```

```
ti=len(U[0])
print "no. of users=",tu   #total no. of users
print "no. of items=",ti   #total no. of items
f=file("D:/myreduceddata60.txt","r")       #file containing userid, movieid,
ratings
d=[line.split() for line in f]#reading file line by line
for it in range(len(d)):
    m=int(d[it][0])
    n=int(d[it][1])
    p=int(d[it][2])
    U[m-1][n-1]=p
genno=18
gentab=[[0]*genno for i in xrange(ti)]
tig=len(U[0])
print "no. of items=",tig   #total no. of users
f=file("D:/final_genre.txt","r") #file containing userid, movieid, ratings
d=[line.split() for line in f]#reading file line by line
for it in range(len(d)):
        mt=int(d[it][0])
        nt=int(d[it][1])
        pt=int(d[it][2])
        qt=int(d[it][3])
        rt=int(d[it][4])
        st=int(d[it][5])
        tt=int(d[it][6])
        ut=int(d[it][7])
        vt=int(d[it][8])
        wt=int(d[it][9])
        xt=int(d[it][10])
        yt=int(d[it][11])
        zt=int(d[it][12])
        at=int(d[it][13])
        bt=int(d[it][14])
        ct=int(d[it][15])
        dt=int(d[it][16])
        et=int(d[it][17])
        ft=int(d[it][18])
        if(nt==1):
            gentab[it][0]=1
        if(pt==1):
            gentab[it][1]=1
        if(qt==1):
            gentab[it][2]=1
        if(rt==1):
            gentab[it][3]=1
        if(st==1):
            gentab[it][4]=1
        if(tt==1):
            gentab[it][5]=1
        if(ut==1):
            gentab[it][6]=1
        if(vt==1):
            gentab[it][7]=1
        if(wt==1):
            gentab[it][8]=1
        if(xt==1):
            gentab[it][9]=1
```

```
        if(yt==1):
            gentab[it][10]=1
        if(zt==1):
            gentab[it][11]=1
        if(at==1):
            gentab[it][12]=1
        if(bt==1):
            gentab[it][13]=1
        if(ct==1):
            gentab[it][14]=1
        if(dt==1):
            gentab[it][15]=1
        if(et==1):
            gentab[it][16]=1
        if(ft==1):
            gentab[it][17]=1
gengr=[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
for i in range(ti):
    for j in range(genno):
        if(gentab[i][j]==1):
            gengr[j].append(i)
groupmat=[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
for i in range(genno):
    d=len(gengr[i])
    if(d>0):
        groupmat[i]=[[0]*d for j in xrange(309)]
for i in range(309):
    for j in range(536):
        if(U[i][j]!=0):
            for k in range(genno):
                if(gengr[k]>0 and j in gengr[k]):
                    val=gengr[k].index(j)
                    #print "val=",val
                    groupmat[k][i][val]=U[i][j]
mistr={}
for k in range(genno):
    c=0
    for i in range(tu):
        for j in range(len(gengr[k])):
            if(groupmat[k][i][j]==0):
                c=c+1
    mistr[k]=c
print "mistr=",mistr

store_mis=[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
for k in range(genno):
    if(mistr[k]>0):
        store_mis[k]=[[0]*3 for i in xrange(mistr[k])]
for k in range(genno):
    if(mistr[k]>0):
        s=0
        for i in range(tu):
            for j in range(len(gengr[k])):
                if(groupmat[k][i][j]==0):
                    store_mis[k][s][0]=i
                    store_mis[k][s][1]=j
                    s=s+1
```

```
def kp(c,d,x,k2):#function to calculate similarity c,x are userids & d item no.
    itemrated=[]
    for j in range(len(gengr[k2])):
        if(d!=j and groupmat[k2][x][j]!=0 and groupmat[k2][c][j]!=0):
            itemrated.append(j)
    s=float(len(itemrated))
    if(len(itemrated)==0):
        return(0)
    if(len(itemrated)!=0):
            sum1=sum([groupmat[k2][c][j] for j in itemrated])
        sum2=sum([groupmat[k2][x][j] for j in itemrated])
        avg1=sum1/s
        avg2=sum2/s
        member1=[]
        member2=[]
        for j in itemrated:
            k=groupmat[k2][c][j]-avg1
            member1.append(k)
        for j in itemrated:
            k=groupmat[k2][x][j]-avg2
            member2.append(k)
        seq=len(member1)
        psum=sum([member1[it]*member2[it] for it in range(seq)])
        if(psum==0.0):
                return(0)
        else:
                sum1sq=sum([pow(member1[it],2) for it in range(seq)])
                sum2sq=sum([pow(member2[it],2) for it in range(seq)])
               den=math.sqrt(sum1sq*sum2sq)
               return(psum/den)

def evaluate(kl,ru,ci):
    userrated=[]
    for i in range(tu):
        if(i!=ru and groupmat[kl][i][ci]!=0):
            userrated.append(i)
        similarity=[]
    for j in range(len(userrated)):
        p=userrated[j]
            sim=kp(ru,ci,p,kl)
        similarity.append(sim)
    avgratings=[]
    for j in range(len(userrated)):
        ks=userrated[j]
        b=0.0
        sumrat=0
        for e in range(len(gengr[kl])):
            if(ci!=e and groupmat[kl][ks][e]!=0):
                b=b+1
                sumrat=sumrat+groupmat[kl][ks][e]
        if(sumrat==0):
            avrat=0.0
        else:
            avrat=sumrat/b
        avgratings.append(avrat)
    ratingitemi=[]
```

```
    for j in range(len(userrated)):
            sr=userrated[j]
            kq=groupmat[kl][sr][ci]
            ratingitemi.append(kq)
    suma=0
    suminp=sum([((ratingitemi[j]-avgratings[j])*similarity[j]) for j in
range(len(userrated))])
    dq=sum([abs(similarity[j]) for j in range(len(userrated))])
    suma=sum([groupmat[kl][ru][j] for j in range(len(gengr[kl])) if
groupmat[kl][ru][j]!=0])
    tt=0.0
    for j in range(len(gengr[kl])):
        if(groupmat[kl][ru][j]!=0):
            tt=tt+1
    if(suma==0):
        avga=0.0
    else:
        avga=suma/tt
    if(dq==0 or suminp==0):
        pre=avga
    else:
        pre=avga+(suminp/dq)
    return(pre)

    for k in range(genno):
    if(mistr[k]>0):
        for it2 in range(mistr[k]):
            rowu=store_mis[k][it2][0]
            coli=store_mis[k][it2][1]
            val=evaluate(k,rowu,coli)
            store_mis[k][it2][2]=val

for k in range(genno):
    a=mistr[k]
    if(a>0):
        for j in range(a):
             c=store_mis[k][j][0]
             d=store_mis[k][j][1]
             b=store_mis[k][j][2]
             groupmat[k][c][d]=b
#print "groupmat with predicted values", groupmat

for i in range(ti):
    groupno=[]
    indx2=[]
    for k in range(genno):
        if(i in gengr[k]):
            groupno.append(k)
            indx2.append(gengr[k].index(i))
    l=float(len(groupno))
    for m in range(tu):
        if(U[m][i]==0.0):
            s=0
            for p in range(len(groupno)):
                q=groupno[p]
                r=indx2[p]
                s=s+groupmat[q][m][r]
```

```
            U[m][i]=round(s/l,5)

f=open('d:/itemval.txt','a')
for i in range(tu):
    for j in range(ti):
        f.write(str(i+1))
        f.write(' ')
        f.write(str(j+1))
        f.write(' ')
        f.write(str(U[i][j]))
        f.write('\n')
f.close()
```

Weighted Slope One Scheme

```
import math
U=[[0]*536 for i in xrange(309)]       #matrix for storing ratings
ti=len(U)
no_items=len(U[0])
print "no. of items=",no_items
print "no. of users=",ti   #total no. of users
f=file("D:/datawithheld.txt","r") #file containing userid, movieid, ratings
d=[line.split() for line in f]#reading file line by line
for it in range(len(d)):
    m=int(d[it][0])
    n=int(d[it][1])
    p=int(d[it][2])
    U[m-1][n-1]=p
stor_rows=no_items*(no_items-1)/2#no. of rows which store preference of items
store=[[0]*4 for i in xrange(stor_rows)] #store matrix created with 4
columns,two for item no's, one for average deviation and one for no. of common
users
k=0   # for filling first column of store
l=0
for i in range(no_items-1,0,-1):
    for j in range(1,i+1,1):
        store[l][0]=k
        l=l+1
    k=k+1

k=1 #for filling second column of store
l=0
x=no_items-3
for i in range(no_items-1,0,-1):
    for j in range(1,i+1,1):
        store[l][1]=k
        l=l+1
        k=k+1
    k=k-x-1
    x=x-1
def avgdif(x,y): #calculates average difference between items(x-y)
    c=0
    sum1=0.0
    sum2=0.0
```

```
    for i in range(ti):
        if(U[i][x]!=0 and U[i][y]!=0):
            c=c+1
            sum1=sum1+U[i][x]
            sum2=sum2+U[i][y]
    if(c==0):
        return(0.0)
    elif((sum1-sum2)==0.0):
        return(0.0)
    else:
        return((sum1-sum2)/c)
def comusers(k,p): #calculates common no. of users
    c=0
    for i in range(ti):
        if(U[i][k]!=0 and U[i][p]!=0):
            c=c+1
    return c
for i in range(stor_rows):#for calculating avg diff & common no. of users for
each item combination stored in store
    k=store[i][0]
    p=store[i][1]
    r=avgdif(k,p)
    g=comusers(k,p)
    store[i][2]=r
    store[i][3]=g
c=0 # for calculating missing values in entire data
for i in range(ti):
    for j in range(no_items):
        if(U[i][j]==0):
            c=c+1
rows_missing=c
missing=[[0]*3 for i in xrange(rows_missing)]
k=0
for i in range(ti):
    for j in range(no_items):
        if(U[i][j]==0):
            missing[k][0]=i
            missing[k][1]=j
            k=k+1
commo=[0]*(no_items-1)
sumrat=[0]*(no_items-1)
finaldev=[0]*(no_items-1)

def calpredict(uno,ino):
    k=0
    for i in range(stor_rows):
        if(store[i][1]==ino or store[i][0]==ino):
                if(store[i][1]==ino):
                    dev=-store[i][2]
                else:
                    dev=store[i][2]
                finaldev[k]=dev
                k=k+1
    j=0
    rat=[0]*(no_items-1)
    for i in range(no_items):
        if(ino!=i):
```

```
            rat[j]=U[uno][i]
            j=j+1
    for i in range(no_items-1):
        sumrat[i]=finaldev[i]+rat[i]
    j=0
    for i in range(stor_rows):
        if((store[i][0]==ino or store[i][1]==ino)):
            commo[j]=store[i][3]
            j=j+1
    den=0
    num=0.0
    for i in range(no_items-1):
        if(rat[i]!=0 and commo[i]!=0):
            den=den+commo[i]
            num=num+commo[i]*sumrat[i]
    if(num==0.0 or den==0):
        return(0.0)
    else:
        return(num/den)
print "missing rows",rows_missing
for i in range(rows_missing):
    u=missing[i][0]
    it=missing[i][1]
    val1=calpredict(u,it)
    print "val1=",val1
    missing[i][2]=round(val1,5)
for i in range(rows_missing):
    x=missing[i][0]
    y=missing[i][1]
    z=missing[i][2]
    U[x][y]=z
f=open('d:/slopevalnew.txt','a')
for i in range(ti):
    for j in range(no_items):
        f.write(str(i+1))
        f.write(' ')
        f.write(str(j+1))
        f.write(' ')
        f.write(str(U[i][j]))
        f.write('\n')
f.close()
```

K-MEANS (for clustering items)

```
import math
U=[[0]*309 for i in xrange(531)]#matrix for storing ratings user-item info
ni1=len(U)
nu1=len(U[0])
print "no. of items=",ni1#total no. of items
print "no. of users=",nu1#total no. of users
f=file("D:/itemdiv_cluster.txt","r")#file containing userid, movieid, ratings
d=[line.split() for line in f]#reading file line by line
for it in range(len(d)):
    m=int(d[it][0])
    n=int(d[it][1])
```

```
    p=float(d[it][2])
    U[m-1][n-1]=p
itemsum=[0]*ni1 # array for storing sum of ratings of each item
for i in range(ni1):
    sum=0
    for j in range(nu1):
        sum=sum+U[i][j]
    itemsum[i]=sum
di={}
for i,v in enumerate(itemsum):
    di[v]=i
diu=di.keys()
diu.sort()
diu.reverse()
k=input("enter the value of k - the no. of clusters to be created")
toprating=[]
for i in range(k):
    h=diu[i]
    j=di[h]
    toprating.append(j)

ltoprat=len(toprating)
temp=[[0]*ni1 for i in xrange(k)]
centroidm=[[0]*nu1 for i in xrange(k)]# centroidm contains the centroids of the
clusters - each row is the cenroid of one cluster
tempc=[[0]*nu1 for i in xrange(k)]

for i in range(k):#to initialize centroid with  ratings of top k rated items
    a=toprating[i]
    for j in range(nu1):
        centroidm[i][j]=U[a][j]
for i in range(k):  # to initialize centroid with ratings of top k rated items
    a=toprating[i]
    for j in range(nu1):
        #print "j=",j
        tempc[i][j]=U[a][j]
def pearson(it1,it2):  # calculating pearson correlation coefficient
     itemrated=[]   # stores index of commonly rated users
    for d in range(nu1):
        if( centroidm[it2][d]!=0 and U[it1][d]!=0):
            itemrated.append(d)
    s=float(len(itemrated))
    if(len(itemrated)==0):
        return(0)
    if(len(itemrated)!=0): # if there are commonly rated items
        sum1=0
        for g in range(len(itemrated)):
            x=itemrated[g]
            sum1=sum1+U[it1][x]
        sum2=0
        for g in range(len(itemrated)):
            x=itemrated[g]
            sum2=sum2+centroidm[it2][x]
        avg1=sum1/s
        avg2=sum2/s
        member1=[]
        member2=[]
```

```
        for t in itemrated:
            ks=(U[it1][t]-avg1)
            member1.append(ks)
        for t in itemrated:
            ks=(centroidm[it2][t]-avg2)
            member2.append(ks)
        seq=len(member1)
        sum=0
        for i in range(seq):
            sum=sum+member1[i]*member2[i]
        psum=sum
        if(psum==0.0):
              return(0)
        else:
            sum1sq=0
            for b in range(seq):
                sum1sq=sum1sq+pow(member1[b],2)
            sum2sq=0
            for b in range(seq):
                sum2sq=sum2sq+pow(member2[b],2)
            den=math.sqrt(sum1sq*sum2sq)
            return(psum/den)

while(True):
      simmat=[[0]*ni1 for i in xrange(k)]
      for i in range(ni1):
         for j in range(k):
               pc=pearson(i,j)
               simmat[j][i]=pc
     findmax=[0]*k
     groupm=[[0]*ni1 for i in xrange(k)]
     for i in range(ni1):
         for j in range(k):
             findmax[j]=simmat[j][i]
         largest=max(findmax)
          indx=findmax.index(largest)
          groupm[indx][i]=1
     for i in range(k):
         eachrowgroup=[]
         for j in range(ni1):
             if(groupm[i][j]==1):
                 eachrowgroup.append(j)
         l=(len(eachrowgroup))
         if(l>0):
             lt=float(l)
             for a in range(nu1):
                 sum=0
                 for b in range(l):
                     x=eachrowgroup[b]
                     sum=sum+U[x][a]
                 centroidm[i][a]=round(sum/lt,4)
      if(temp==groupm and centroidm==tempc):
              break
     temp=groupm
     tempc=centroidm
print "len of centroid 0 and 1",len(centroidm[0]),"and",len(centroidm[1])
c=0
```

```
for i in range(k):
    for j in range(ni1):
        if(groupm[i][j]==1):
            c=c+1
print "total items", c
f=open('d:/slopegrpf.txt','a')
for i in range(k):
    for j in range(ni1):
        f.write(str(i+1))
        f.write(' ')
        f.write(str(j+1))
        f.write(' ')
        f.write(str(groupm[i][j]))
        f.write('\n')
f.close()
f=open('d:/slopecentrf.txt','a')
for i in range(k):
    for j in range(nu1):
        f.write(str(i+1))
        f.write(' ')
        f.write(str(j+1))
        f.write(' ')
        f.write(str(centroidm[i][j]))
        f.write('\n')
f.close()
```

Selecting clusters , finding neighbors and predicting value

```
k=48
import math
U=[[0]*536 for i in xrange(309)]#matrix for storing user-item info
nu1=len(U)
ni1=len(U[0])
print "no. of items=",ni1#total no. of items
print "no. of users=",nu1#total no. of users
f=file("D:/itemnewop.txt","r")       #file containing userid, movieid, ratings
d=[line.split() for line in f]#reading file line by line
for it in range(len(d)):
    m=int(d[it][0])
    n=int(d[it][1])
    p=float(d[it][2])
    U[m-1][n-1]=p

itemclustermat=[[0]*309 for i in xrange(531)]
nu_cluster1=len(itemclustermat)
ni_cluster1=len(itemclustermat[0])
print "no. of users in div cluster file=",ni_cluster1#total no. of users
print "no. of items in div cluster file=",nu_cluster1#total no. of items
f=file("D:/itemdiv_cluster.txt","r")#file containing userid, movieid, ratings
d=[line.split() for line in f]#reading file line by line
for it in range(len(d)):
    m=int(d[it][0])
    n=int(d[it][1])
    p=float(d[it][2])
    itemclustermat[m-1][n-1]=p
```

```
no_itmclstr=531
groupm=[[0]*no_itmclstr for i in xrange(k)]
f=file("D:/itemgrpf.txt","r")       #file containing userid, movieid, ratings
d=[line.split() for line in f]#reading file line by line
for it in range(len(d)):
    m=int(d[it][0])
    n=int(d[it][1])
    p=int(d[it][2])
    groupm[m-1][n-1]=p
centroidm=[[0]*nu1 for i in xrange(k)]
f=file("D:/itemcentrf.txt","r")       #file containing userid, movieid, ratings
d=[line.split() for line in f]#reading file line by line
for it in range(len(d)):
    m=int(d[it][0])
    n=int(d[it][1])
    p=float(d[it][2])
    centroidm[m-1][n-1]=p

tgtitm=[]
f=file("D:/targetitm2.txt","r")       #file containing userid, movieid, ratings
d=[line.split() for line in f]#reading file line by line
for it in range(len(d)):
    p=float(d[it][2])
    tgtitm.append(p)

def pearson(it1):  # calculating pearson correlation coefficient
    itemrated=[]   # stores index of commonly rated users
    for d in range(nu1):
        if( centroidm[it1][d]!=0 and tgtitm[d]!=0):
            itemrated.append(d)
    s=float(len(itemrated))
    if(len(itemrated)==0):
        return(0)
   if(len(itemrated)!=0): # if there are commonly rated items
        sum1=0
        for g in range(len(itemrated)):
            x=itemrated[g]
            sum1=sum1+tgtitm[x]
        sum2=0
        for g in range(len(itemrated)):
            x=itemrated[g]
            sum2=sum2+centroidm[it1][x]
        avg1=sum1/s
        avg2=sum2/s
        member1=[]
        member2=[]
        for t in itemrated:
            ks=(tgtitm[t]-avg1)
            member1.append(ks)
        for t in itemrated:
            ks=(centroidm[it1][t]-avg2)
            member2.append(ks)
        seq=len(member1)
        sum=0
        for i in range(seq):
            sum=sum+member1[i]*member2[i]
        psum=sum
```

```
        if(psum==0.0):
            return(0)
        else:
            sum1sq=0
            for b in range(seq):
                sum1sq=sum1sq+pow(member1[b],2)
            sum2sq=0
            for b in range(seq):
                sum2sq=sum2sq+pow(member2[b],2)
            den=math.sqrt(sum1sq*sum2sq)
            return(psum/den)
sim=[]
for i in range(k):
    p=pearson(i)
    sim.append(p)
neigh_grp=[]
for i in range(k):
    c=0
    for j in range(no_itmclstr):
        if(groupm[i][j]==1):
            c=c+1
    neigh_grp.append(c)
itemno_neigh=[[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]
,[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
for i in range(k):
    for j in range(no_itmclstr):
        if(groupm[i][j]==1):
            itemno_neigh[i].append(j)

def cosinesim(it1):# calculating cosine similarity
    print "for item no.",it1
    itemrated=[]   # stores index of commonly rated users
    for d in range(nu1):
        if( itemclustermat[it1][d]!=0 and tgtitm[d]!=0):
            itemrated.append(d)
    s=float(len(itemrated))
    if(len(itemrated)==0):
        return(0)
    if(len(itemrated)!=0): # if there are commonly rated items
        member1=[]
        member2=[]
        for t in itemrated:
            ks=tgtitm[t]
            member1.append(ks)
        for t in itemrated:
            ks=itemclustermat[it1][t]
            member2.append(ks)
        seq=len(member1)
        sum=0
        for i in range(seq):
            sum=sum+member1[i]*member2[i]
        psum=sum
        if(psum==0.0):
            return(0)
        else:
            sum1sq=0
            for b in range(seq):
```

```
                sum1sq=sum1sq+pow(member1[b],2)
            sum2sq=0
            for b in range(seq):
                sum2sq=sum2sq+pow(member2[b],2)
            den=math.sqrt(sum1sq*sum2sq)
            return(psum/den)

def completegr(gno,no):
    neighborlist=[]
    neighborlist.extend(itemno_neigh[gno])
    cossim=[]
    for j in neighborlist:
        val=cosinesim(j)
        cossim.append(val)
    print "similarity with the neighbors",cossim
    copycossim=[]
    copycossim.extend(cossim)
    copycossim.sort()
    copycossim.reverse()
    finalsim1=[]
    for i in range(no):
        finalsim1.append(copycossim[i])
    finalneighbor1=[]
    for i in range(no):
        st=finalsim1[i]
        indx3=cossim.index(st)
        finalneighbor1.append(neighborlist[indx3])
    uno=input("enter the user number")
    userno=uno-1
    den=sum([finalsim1[b] for b in range(no)])
    den1=float(den)
    print "denominator=",den1
    a1=0
    sum1=0
    for j in finalneighbor1:
        sum1=sum1+itemclustermat[j][userno]*finalsim1[a1]
        a1=a1+1
    psum=sum1
    if(psum==0):
        return(0)
    else:
        return(psum/den1)

number=input("enter the number of neighbors ")
copysim=[]
copysim.extend(sim)
copysim.sort()
copysim.reverse()
indexlist=[]
for j in copysim:
    indx=sim.index(j)
    indexlist.append(indx)
k=copysim[0]
indx=sim.index(k)
print "maximum similarity with cluster",indx
imm_neigh=neigh_grp[indx]
if(number <= imm_neigh):
```

```
    print " we can find all neighbors in a single cluster"
    val5=completegr(indx,number)
    print "predicted value=",val5
else:
    print "we need to look multiple clusters"
    oldsum=0
    sum5=0
    all=[]
    clusterno=[]
    neighlist3=[]
    for j in copysim:
        indx7=sim.index(j)
        clusterno.append(indx7)
        ngh=neigh_grp[indx7]
        sum5=sum5+ngh
        if(sum5>number):
            count=number-oldsum
            neighlist3.append(count)
            all.append(0)
            break
        if(sum5<number):
            neighlist3.append(ngh)
            all.append(1)
        if(sum5==number):
            neighlist3.append(ngh)
            all.append(1)
            break
        oldsum=sum5
    f=1
    for i in range(len(all)):
        if(all[i]==0):
            f=0
            break
    if(f==1):
        print "the neighbors are found in complete group"
        finalneighbor6=[]
        for i in range(len(clusterno)):
            cno=clusterno[i]
            finalneighbor6.extend(itemno_neigh[cno])
        finalsim6=[]
        for j in finalneighbor6:
            valu3=cosinesim(j)
            finalsim6.append(valu3)
        uno1=input("enter the user number")
        userno1=uno1-1
        den3=sum([finalsim6[b] for b in range(number)])
        den2=float(den3)
        a2=0
        sum2=0
        for j in finalneighbor6:
            sum2=sum2+itemclustermat[j][userno1]*finalsim6[a2]
            a2=a2+1
        psum1=sum2
        if(psum1==0):
            print "predicted value =",0
        else:
            print "predicted value=",psum1/den2
```

```
else:
    print "the neighbors are not in complete group"
    len6=len(clusterno)
    len6=len6-1
    v1=neighlist3[len6]
    cno1=clusterno[len6]
    inn2=[]
    inn2.extend(itemno_neigh[cno1])
    similar5=[]
    for j in inn2:
        v3=cosinesim(j)
        similar5.append(v3)
    copysimilar5=[]
    copysimilar5.extend(similar5)
    copysimilar5.sort()
    copysimilar5.reverse()
    indlist3=[]
    for j in copysimilar5:
        val=similar5.index(j)
        its=inn2[val]
        indlist3.append(its)
    sim_5=[]
    inn_5=[]
    for i in range(v1):
        sim_5.append(copysimilar5[i])
        indx_5=similar5.index(copysimilar5[i])
        inn_5.append(inn2[indx_5])
    print "top similar neighbors in the last cluster",sim_5
    fn1=[]
    fsim1=[]
    for i in range(len(clusterno)-1):
        clno=clusterno[i]
        fn1.extend(itemno_neigh[clno])
    for j in fn1:
        v2=cosinesim(j)
        fsim1.append(v2)
    fn1.extend(inn_5)
    fsim1.extend(sim_5)
    le2=len(fsim1)
    uno3=input("enter the user number")
    userno3=uno3-1
    den4=sum([fsim1[b] for b in range(le2)])
    den4=float(den4)
    i=0
    sum_3=0
    for j in fn1:
        sum_3=sum_3+itemclustermat[j][userno3]*fsim1[i]
        i=i+1
    psum=sum_3
    if(psum==0):
        print "predicted value =",0
    else:
        print "predicted value=",psum/den4
```

APPENDIX B

APPENDIX B

RESULTS

Using Item Classification Technique and Item clustering

No. of neighbors=6

Pre rat : predicted rating

Actual rat : actual rating

Pre rat.	Actual rat.	Pre rat.	Actual rat.	Pre rat.	Actual rat.	Pre rat.	Actual rat.	Pre rat.	Actual rat.
4.447092	4	4.054278	4	3.885695	5	2.878094	4	3.579958	4
4.679484	4	4.841588	5	4.157742	5	3.455908	2	3.669238	4
3.5004	3	2.62092	3	3.628885	5	2.692162	2	2.844484	3
3.898149	5	2.84753	2	3.676934	5	3.022588	3	3.502278	3
4.420494	4	3.925422	5	4.268989	5	3.41981	4	2.901473	3
3.877591	4	3.510301	5	4.311304	5	3.2769	3	3.379595	4
3.24533	4	3.819959	4	3.543191	5	2.832477	4	3.19495	4
3.160002	4	2.959836	2	4.152219	3	3.672887	3	3.71173	4
3.822456	2	3.416087	3	3.996633	5	2.806284	3	3.071239	4
4.267351	4	4.123496	5	4.333419	5	3.39399	3	3.702737	4
3.644186	2	4.050412	4	4.335038	5	3.548201	3	3.762214	5
3.499133	3	3.139822	3	2.551045	4	2.662468	2	2.382305	3
4.337399	5	3.722277	5	4.521594	5	3.828206	4	3.693621	4
3.650867	5	3.949098	3	3.464083	5	3.173247	2	3.141476	4
3.815945	4	3.653458	3	3.864616	4	2.807937	3	3.065456	3

MAE=.696975

No. of neighbors=8

Pre rat	Actual Rat	Pre rat	Actual Rat	Pre rat	Actual Rat	Pre rat	Actual Rat	Pre rat	Actual Rat
4.335198	4	3.915892	4	3.937885	5	3.013178	4	3.472693	4
4.474884	4	4.683764	5	4.165482	5	3.464099	2	3.702637	4
3.363273	3	2.71557	3	3.60163	5	2.634489	2	2.823474	3
3.673369	5	2.760751	2	3.534525	5	2.767762	3	3.5478	3
4.143456	4	3.944042	5	4.077795	5	3.303652	4	3.021936	3
3.701992	4	3.322164	5	4.198992	5	3.241337	3	3.343934	4
3.234777	4	3.864912	4	3.599708	5	2.825921	4	3.138759	4
3.245184	4	2.969864	2	4.034434	3	3.501997	3	3.74149	4
3.695854	2	3.356011	3	3.836742	5	2.801672	3	3.063932	4
4.191454	4	4.092661	5	4.124648	5	3.416913	3	3.776999	4
3.581285	2	3.90164	4	4.364358	5	3.661061	3	3.727711	5
3.374215	3	3.033011	3	2.663376	4	2.635604	2	2.339217	3
4.503228	5	3.541937	5	4.502431	5	3.763466	4	3.736615	4
3.67426	5	3.879515	3	3.507617	5	2.965109	2	3.142633	4
3.542094	4	3.518941	3	3.728908	4	2.896521	3	3.1145	3

MAE=.695108

No. of neighbors=10

Pre rat	Actual Rat	Pre rat	Actual Rat	Pre rat	Actual Rat	Pre rat	Actual Rat	Pre rat	Actual Rat
4.068257	4	3.832839	4	4.036499	5	3.309323	4	3.526053	4
4.335976	4	4.547219	5	4.258473	5	3.354504	2	3.574577	4
3.277685	3	2.794074	3	3.567731	5	2.709719	2	2.711576	3
3.62906	5	2.692894	2	3.509647	5	2.836201	3	3.438301	3
4.114787	4	3.931921	5	4.069137	5	3.456003	4	3.11529	3
3.596341	4	3.235183	5	4.103136	5	3.209319	3	3.31885	4
3.320178	4	3.79204	4	3.750854	5	2.891433	4	3.243205	4
3.296132	4	3.015283	2	3.993116	3	3.431215	3	3.693245	4
3.756658	2	3.335049	3	3.835857	5	2.841183	3	3.089495	4
4.038703	4	4.074156	5	4.112019	5	3.533075	3	3.921495	4
3.620959	2	3.950831	4	4.340137	5	3.645465	3	3.770655	5
3.376464	3	2.929585	3	2.830657	4	2.807627	2	2.422582	3
4.502577	5	3.533565	5	4.380533	5	3.801346	4	3.780689	4
3.539473	5	3.89556	3	3.523667	5	3.074989	2	3.15701	4
3.452334	4	3.416299	3	3.74197	4	2.963148	3	3.123261	3

MAE=.680727

No. of neighbors=12

Pre rat	Actual Rat	Pre rat	Actual Rat	Pre rat	Actual Rat	Pre rat	Actual Rat	Pre rat	Actual Rat
3.973061	4	3.844572	4	4.102947	5	3.423854	4	3.658781	4
4.229482	4	4.553281	5	4.216528	5	3.405385	2	3.624056	4
3.194402	3	2.888357	3	3.564934	5	2.69253	2	2.76184	3
3.570785	5	2.811287	2	3.543239	5	2.855966	3	3.448555	3
4.114227	4	3.94325	5	4.051654	5	3.564941	4	3.234183	3
3.547382	4	3.229883	5	4.050414	5	3.231358	3	3.291039	4
3.313371	4	3.906603	4	3.755681	5	2.95754	4	3.285941	4
3.306233	4	3.137846	2	3.95812	3	3.410185	3	3.744281	4
3.630354	2	3.37739	3	3.827469	5	2.935601	3	3.100524	4
4.020474	4	4.113163	5	4.176648	5	3.625398	3	3.917751	4
3.649501	2	4.019995	4	4.292911	5	3.675735	3	3.625092	5
3.283913	3	2.898365	3	2.858867	4	2.797099	2	2.475824	3
4.499788	5	3.6944	5	4.329475	5	3.917308	4	3.786525	4
3.467657	5	3.922913	3	3.556291	5	3.076519	2	3.155166	4
3.488071	4	3.405549	3	3.740341	4	3.010493	3	3.151067	3

MAE=.66835

No. of neighbors=14

Pre rat	Actual Rat	Pre rat	Actual Rat	Pre rat	Actual Rat	Pre rat	Actual Rat	Pre rat	Actual Rat
3.833869	4	3.830227	4	4.081776	5	3.576742	4	3.684282	4
4.14413	4	4.489477	5	4.263272	5	3.380211	2	3.611204	4
3.139001	3	2.885785	3	3.490232	5	2.749167	2	2.800681	3
3.489014	5	2.670655	2	3.425559	5	2.939549	3	3.486877	3
4.097922	4	3.951341	5	4.016079	5	3.634609	4	3.05812	3
3.473109	4	3.168981	5	4.01556	5	3.170005	3	3.249522	4
3.195774	4	3.848611	4	3.790539	5	3.105641	4	3.339558	4
3.190707	4	3.118193	2	3.925154	3	3.38865	3	3.758811	4
3.548625	2	3.394846	3	3.835338	5	3.039032	3	3.142645	4
3.985325	4	4.087825	5	4.294121	5	3.676064	3	3.917036	4
3.651974	2	4.018196	4	4.254207	5	3.650771	3	3.653988	5
3.199558	3	2.859738	3	2.80763	4	2.92069	2	2.479276	3
4.41917	5	3.737968	5	4.286348	5	3.930492	4	3.828606	4
3.472421	5	3.915036	3	3.541292	5	3.148366	2	3.157012	4
3.489922	4	3.336201	3	3.711835	4	3.060311	3	3.157701	3

MAE=.666836

Using Weighted Slope one scheme and Item clustering

No. of neighbors=6

Pre rat : predicted rating

Actual rat : actual rating

Pre Rat	Actual Rat	Pre Rat	Actual Rat	Pre Rat	Actual Rat	Pre Rat	Actual Rat	Pre Rat	Actual Rat
3.771887	4	4.252483	4	3.703456	5	3.478749	4	3.527947	4
3.978097	4	4.538225	5	4.109109	5	3.70761	2	3.519338	4
3.131967	3	3.714024	3	3.540758	5	2.948917	2	2.817101	3
3.552932	5	4.024951	2	3.854411	5	4.149563	3	3.611031	3
4.301203	4	4.406705	5	4.423963	5	3.79972	4	3.649225	3
3.72715	4	4.436714	5	4.071103	5	3.666714	3	3.833281	4
3.625225	4	4.038809	4	3.552864	5	3.399332	4	3.308322	4
3.744442	4	3.913577	2	4.06821	3	3.556006	3	3.506115	4
3.594493	2	3.815497	3	3.958245	5	3.492522	3	3.574611	4
4.017775	4	3.698761	5	3.998824	5	4.111313	3	4.058118	4
3.857931	2	4.295343	4	4.271942	5	3.527524	3	3.630347	5
3.088844	3	3.147182	3	2.571817	4	2.99972	2	3.166497	3
4.693009	5	4.488164	5	4.584289	5	4.241981	4	4.075443	4
3.728595	5	3.936152	3	3.524587	5	3.51772	2	3.379486	4
3.305931	4	3.716047	3	3.816923	4	3.133352	3	3.074971	3

MAE=.708669

No. of neighbors=8

Pre Rat	Actual Rat	Pre Rat	Actual Rat	Pre Rat	Actual Rat	Pre Rat	Actual Rat	Pre Rat	Actual Rat
3.703601	4	4.315012	4	3.883844	5	3.484033	4	3.484254	4
3.898916	4	4.357725	5	4.072442	5	3.574562	2	3.574309	4
3.054048	3	3.785422	3	3.436204	5	2.983346	2	2.982975	3
3.516946	5	4.093289	2	3.815426	5	3.958757	3	3.957674	3
4.051593	4	4.478963	5	4.322711	5	3.737091	4	3.736767	3
3.572335	4	4.327682	5	4.13028	5	3.749839	3	3.750212	4
3.625217	4	4.102694	4	3.591047	5	3.299691	4	3.299667	4
3.579386	4	3.966879	2	4.05112	3	3.546639	3	3.546633	4
3.57056	2	4.111225	3	3.882704	5	3.49437	3	3.494758	4
3.888625	4	4.023633	5	3.999119	5	4.030432	3	4.030395	4
3.808686	2	4.358288	4	4.187844	5	3.571487	3	3.571784	5
3.130459	3	3.172939	3	2.679094	4	2.99979	2	3.000269	3
4.612352	5	4.4522	5	4.485946	5	4.181602	4	4.181291	4
3.44015	5	3.998062	3	3.544169	5	3.341649	2	3.341661	4
3.35476	4	3.922198	3	3.745715	4	3.100078	3	3.10001	3

MAE=.706247

No. of neighbors=10

Pre Rat	Actual Rat	Pre Rat	Actual Rat	Pre Rat	Actual Rat	Pre Rat	Actual Rat	Pre Rat	Actual Rat
3.462974	4	4.309424	4	3.864234	5	3.08843	4	3.387702	4
3.951708	4	4.406828	5	4.038894	5	3.564659	2	3.542112	4
2.97874	3	3.728151	3	3.348629	5	2.986666	2	2.968527	3
3.3576	5	4.090273	2	3.76067	5	3.767609	3	4.165496	3
4.041289	4	4.47952	5	4.239033	5	3.689776	4	3.689528	3
3.503554	4	4.274373	5	4.012161	5	3.600343	3	3.800016	4
3.549137	4	4.099319	4	3.63059	5	3.14021	4	3.339578	4
3.26346	4	3.958932	2	3.986226	3	3.457941	3	3.73675	4
3.497672	2	4.076934	3	3.816362	5	3.395807	3	3.528224	4
3.953955	4	4.018914	5	4.099377	5	4.024365	3	4.024335	4
3.752655	2	4.356348	4	4.116875	5	3.557189	3	3.62437	5
3.005521	3	3.255305	3	2.743378	4	2.900101	2	2.900492	3
4.44096	5	4.361906	5	4.402109	5	4.045757	4	4.244867	4
3.551961	5	4.079673	3	3.534676	5	3.388505	2	3.320226	4
3.284408	4	3.921857	3	3.681965	4	2.980394	3	3.132776	3

MAE=.694125

No. of neighbors=12

Pre Rat	Actual Rat	Pre Rat	Actual Rat	Pre Rat	Actual Rat	Pre Rat	Actual Rat	Pre Rat	Actual Rat
3.386099	4	4.269697	4	3.978585	5	3.073751	4	3.074385	4
3.8471	4	4.505508	5	4.115757	5	3.539456	2	3.539254	4
2.897836	3	3.523762	3	3.457143	5	2.974024	2	2.973749	3
3.298222	5	4.050712	2	3.71593	5	3.972184	3	3.972085	3
4.034433	4	4.495763	5	4.144002	5	3.658249	4	3.658019	3
3.391939	4	4.200146	5	4.0018	5	3.666686	3	3.667255	4
3.349545	4	4.081877	4	3.775474	5	3.19991	4	3.200195	4
3.302738	4	3.799551	2	3.98852	3	3.630946	3	3.631369	4
3.415033	2	3.981029	3	3.7574	5	3.440096	3	3.440566	4
3.860164	4	4.098873	5	4.191076	5	4.02032	3	4.020296	4
3.709887	2	4.316882	4	4.263998	5	3.603393	3	3.603674	5
2.886108	3	3.233976	3	2.786129	4	2.83366	2	2.833973	3
4.294451	5	4.354037	5	4.41837	5	4.121185	4	4.121316	4
3.460308	5	4.026938	3	3.601359	5	3.362883	2	3.362798	4
3.237181	4	3.884391	3	3.651697	4	3.027531	3	3.027711	3

MAE=.683655

No. of neighbors=14

Pre Rat	Actual Rat	Pre Rat	Actual Rat	Pre Rat	Actual Rat	Pre Rat	Actual Rat	Pre Rat	Actual Rat
3.381724	4	4.253851	4	4.124275	5	2.992111	4	3.063806	4
3.742707	4	4.576091	5	4.08076	5	3.502482	2	3.592864	4
2.770979	3	3.412386	3	3.456174	5	2.83542	2	2.977485	3
3.154806	5	3.834707	2	3.720253	5	3.833842	3	3.976058	3
3.887236	4	4.461993	5	4.136496	5	3.635746	4	3.848878	3
3.232845	4	4.137024	5	3.957077	5	3.571816	3	3.643469	4
3.276224	4	4.041372	4	3.8075	5	3.242619	4	3.313944	4
3.116627	4	3.676236	2	3.906634	3	3.612307	3	3.541575	4
3.395017	2	3.912696	3	3.75151	5	3.415822	3	3.52013	4
3.593492	4	4.156459	5	4.23517	5	3.946284	3	4.088522	4
3.625389	2	4.279311	4	4.31395	5	3.588685	3	3.67101	5
2.785303	3	3.223541	3	2.745347	4	2.703509	2	2.786476	3
4.156938	5	4.267427	5	4.385289	5	4.175096	4	4.246284	4
3.355834	5	4.020391	3	3.602965	5	3.240111	2	3.453422	4
3.061283	4	3.968646	3	3.701378	4	2.997723	3	3.040992	3

MAE=.678434